RasPi

DESIGN
BUILD
CODE

21

Get hands-on with your Raspberry Pi

CODE A SIMPLE SYNTH

TOP 10 **ANDROID APPS**

Plus

Run science experiments on your Pi

Welcome

Did you know that your Raspberry Pi is powerful enough to handle some pretty high-level audio processing? This month we're going to show you how to code a simple polyphonic synthesiser using Python, from scratch. It's such a worthwhile project because the lessons you'll learn while working with buffers, manipulating the flow of data into each of them, and running the maths on the outputs, will really help you out with your audio, video and electronics projects. As well as this, we're also playing around with a great little device for doing science experiments, and following up last issue's guide to profiling your Python code with a tutorial on optimisation. Enjoy!

Gavin Thomas

Editor

Get inspired

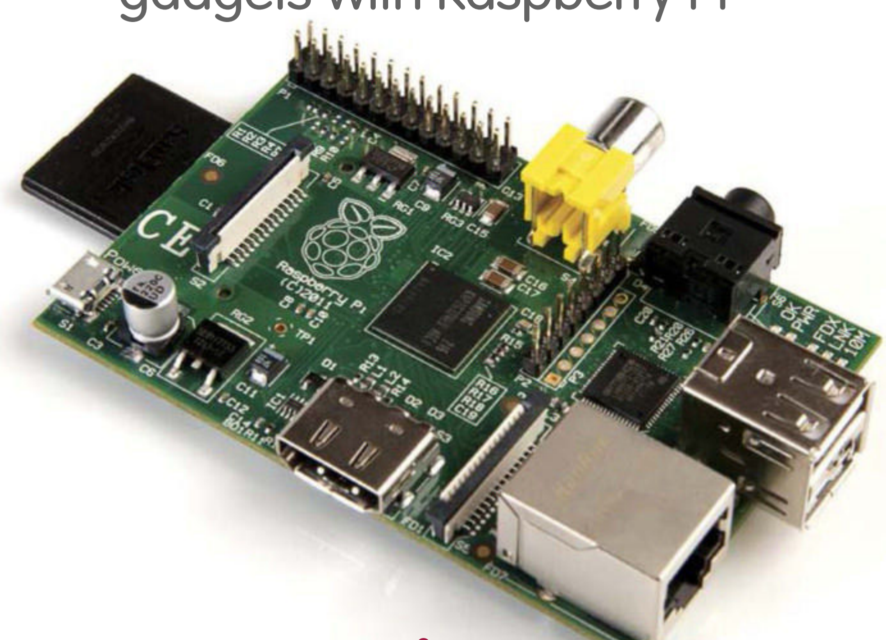
Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi



From the makers of LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk

Contents

Code a simple synthesiser

Learn how to write a simple polyphonic synthesiser (and the theory behind it) using Python and Cython

We are going to take you through the basics of wavetable synthesis theory and use that knowledge to create a real-time synthesiser in Python. At the moment, it is controlled by the computer keyboard, but it could easily be adapted to accept a MIDI keyboard as input.

The Python implementation of such a synthesiser turns out to be too slow for polyphonic sound (ie playing multiple notes at the same time) so we'll use Cython, which compiles Python to C so that you can then compile it to native machine code to improve the performance. The end result is polyphony of three notes, so this is not intended for use as a serious synthesiser. Instead, this tutorial will enable you to become familiar with synthesis concepts in a comfortable language: Python.

Once you're finished, try taking this project further by customising the key mapping to better fit your keyboard layout, or tweaking the code in order to read input from a MIDI keyboard.

THE PROJECT ESSENTIALS

Raspberry Pi 2 or 3
USB sound card

01 Install packages

Using the latest Raspbian image, install the required packages with the following commands:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python-pip python2.7-dev
    portaudio19-dev
sudo pip install cython pyaudio
```

The final step compiles Cython and PyAudio from source, so you might want to go and do something else while it works its magic.

"We'll use Cython, which compiles Python to C so that you can then compile it to native machine code to improve the performance"

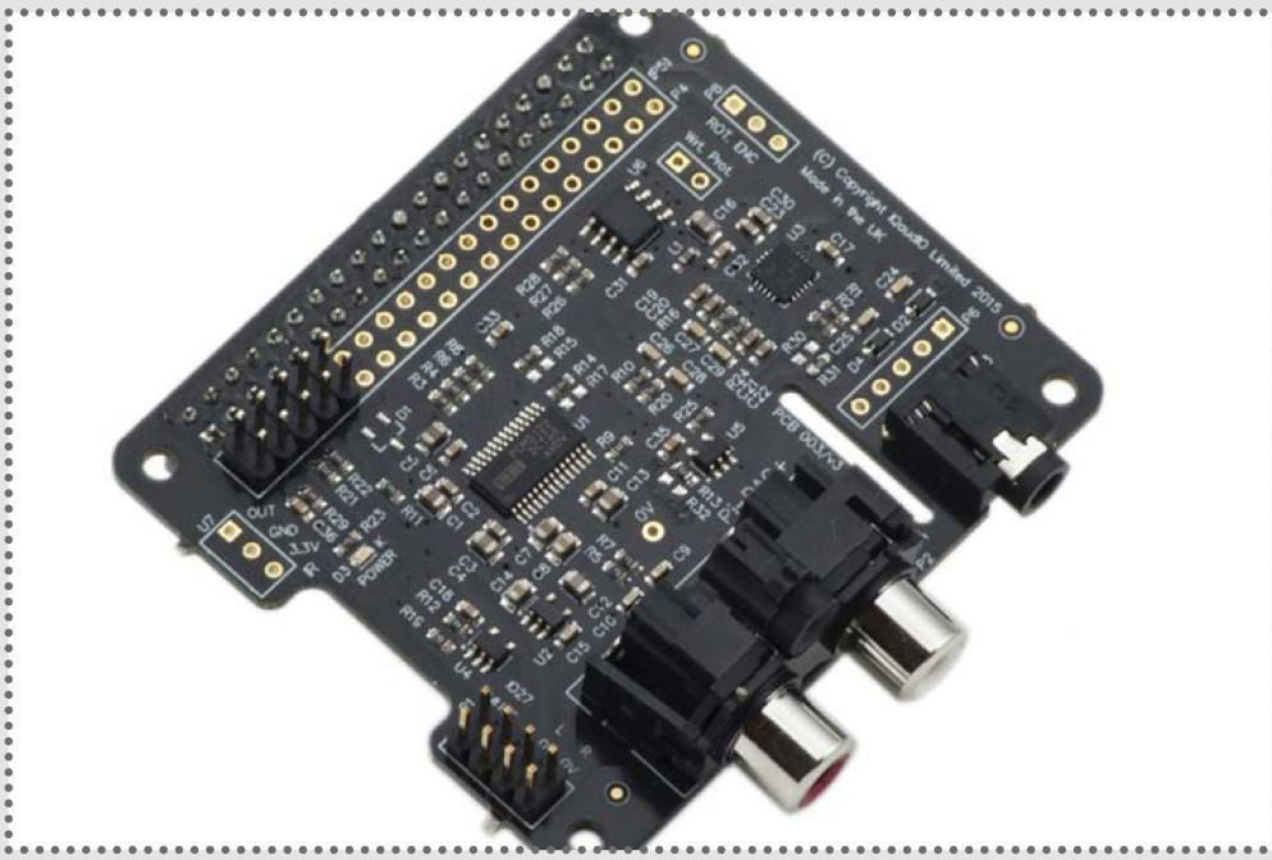
02 Disable built-in sound card

We had issues getting the Raspberry Pi's built-in sound card to work reliably while developing the synthesiser code. For that reason, we are using a USB sound card and will disable the built-in card so that the default card is the USB one:

```
sudo rm /etc/modprobe.d/alsa*
sudo editor /etc/modules
```

Comment out the relevant line by changing 'snd-bcm2835' to '#snd-bcm2835' and save, then:

```
sudo reboot
```



03 Test sound card

Now we can test the USB sound card. Type **alsamixer** and then ensure that the volume is set to a comfortable level. If you're plugging speakers in, you'll probably want it set to 100%. Then type **speaker-test**, which will generate some pink noise on the speakers. Press Ctrl+C to exit once you are happy that it's working.

Cython

Cython compiles Python down to the C code that would be used by the interpreter to run the code. This can optimise some parts of your Python code into pure C code, which is significantly faster, by giving C types such as int, float and char to Python variables. Once you have C code, it can be compiled with a C compiler (usually GCC) that optimises the code even further. For more details, go to **www.cython.org**.

04 Start project

Start by creating a directory for the project. Then download one cycle of a square wave that we will use as a wavetable, like so:

```
mkdir synth
cd synth
wget liamfraser.co.uk/lud/synth/square.wav
```

05 Create compilation script

We need a script that will profile our Python code (resulting in synth.html). First generate a Cython code for it, and then compile the Cython code to a binary using GCC:

```
editor compile.sh:
#!/bin/bash
cython -a synth.pyx
cython --embed synth.pyx
gcc -march=armv7-a -mfpu=neon-vfpv4 -mfloat-
    abi=hard -O3 -I /usr/include/python2.7 -o
    synth.bin synth.c -lpython2.7 -lpthread
```

(Notice the options that tell the compiler to use the floating point unit.) Make it executable with:

```
chmod +x compile.sh
```

06 Start to code

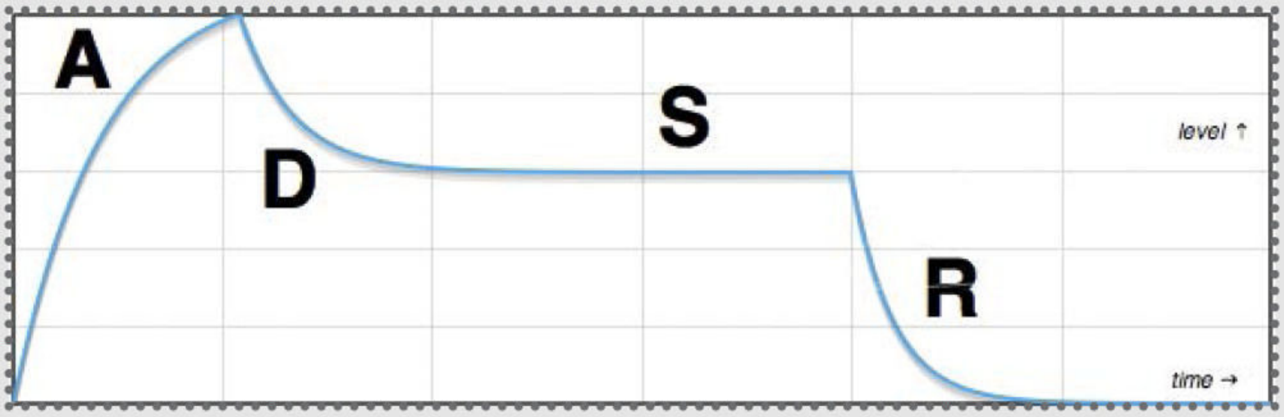
Our code file is going to be called synth.pyx. This extension tells Cython that it is not plain Python code – and as such, can't be run in a normal Python interpreter. Create the file with your favourite editor and add the imports.

07 MIDI Table

To synthesise the standard note of a piano, we need a

"This extension tells Cython that it is not plain Python code – and as such, can't be run in a normal Python interpreter"

table of MIDI values. MIDI notes range from 0-127. MIDI note 60 is middle C on a piano. The MIDI Table class has a 'get note' function that returns the frequency of a note when you give it a MIDI note number.



08 Attack, Decay, Sustain, Release

The ADSR class applies a volume curve over time to the raw output of an oscillator. It does this by returning a multiplier to the note that is a multiple between 0.0 and 1.0. The version we provide has an attack time of 100 ms, a decay time of 300 ms and a release time of 50 ms. You can try changing these values to see how it affects the sound.

The ADSR class does a lot of maths (44,100 times per second, per note). As such, we want to give types to all of the variables so that the maths can be optimised into a raw C loop where possible, because Python has a massive amount of overhead when compared to C. This is what the **cdef** keyword does. If **cdef public** is used, then the variable can also be accessed from inside Python as well.

09 Generate notes

The note class is the core of our synthesiser. It uses the wavetable to generate waves of a specific frequency. The synthesiser asks the note class for a sample. After

"To synthesise the standard note of a piano, we need a table of MIDI values. MIDI notes range from 0-127. MIDI note 60 is middle C on a piano"

generating a sample, the ADSR multiplier is applied and then returned to the synthesiser. If you're interested in the maths behind this, it's worth having a read of the Wikipedia page: **http://bit.ly/1KgI9dp**.

The **note** class does as much maths as the ADSR class, so it is optimised as much as possible using **cdef** keywords. The **cpdef** keyword used for the **next_sample** function means that the function can be called from a non-cdef class. However, the main **synth** class is much too complicated to give static types to absolutely everything.

10 The audio flow

This **synth** class is the main class of the application. It has two sample buffers that are the length of the buffer size. While one buffer is being played by the sound card, the other buffer is being filled in a different thread. Once the sound card has played a buffer, the callback function is called. References to the buffers are swapped and the buffer that has just been filled is returned to the audio library.

The smaller the buffer size, the lower the latency. The Raspbian image isn't optimised for real time audio by default so you may have trouble getting small buffer sizes. It also depends on the USB sound card used.

11 Synth loop

The **start** method of the synth class initialises the audio hardware and then starts the **synth_loop** method in its own thread. While the **exit** event is set to false, the **do_sample** function is called.

The do_sample function loops through the notes that are currently turned on and asks for a sample from

"The synth asks the note class for a sample. After generating a sample, the ADSR multiplier is applied and then returned to the synthesiser"

each one. These samples are shifted right by three (ie divided by 2^3) and added to **out_sample**. The division ensures that the output sample can't overflow (this is a very primitive method of adding notes together, but it works nonetheless).

The resulting sample is then put in the sample buffer. Once the buffer is full, the **more_samples** condition is cleared and the **synth_loop** thread waits to be notified that the buffer it has just built has been sent to the audio card. At this point, the synthesiser can fill up the buffer that has just finished playing and then the cycle continues.

12 Turn on notes

There are both **note_on/off** and **freq_on/off** functions that enable either MIDI notes or arbitrary frequencies to be turned on easily. Added to this, there is also a **toggle note** function which keeps track of MIDI notes that are on and turns them off if they are already on. The toggle note method is used specifically for keyboard input.

13 Add keyboard input

For keyboard input, we needed the ability to get a single character press from the screen. Python's usual input code needs entering before returning to the program. Our code for this is inspired by: **https://code.activestate.com/recipes/577977-get-single-keypress**.

There is a mapping of letters on a keyboard to MIDI note numbers for an entire keyboard octave. We have tried to match the letter spacing to how a piano is laid out to make things easier. However, more innovative methods of input are left as an exercise to the reader.

"There is a mapping of letters on a keyboard to MIDI note numbers for an entire keyboard octave. We have tried to match the letter spacing to how a piano is laid out to make things easier"

14 Put it all together

The **main** function of the program creates an instance of the synth class and then starts the audio stream and synth loop thread. The start function will then return control to the main thread again.

At this point we create an instance of the KB input class and enter a loop that gets characters and toggles the corresponding MIDI note on or off. If the user presses the Q key, that will stop the synth and end the input loop. The program will then exit.

15 Compile the code

Exit your editor and run the compile script by typing the following command:

```
./compile.sh
```

This may take around 30 seconds, so don't worry if it isn't instant. Once the compilation has finished, execute the synth.bin command using:

```
./synth.bin
```

Pressing keys from A all the way up to K on the keyboard will emulate the white keys on the piano. If you press a key again the note will go off successfully.

```
note 60 on
Note: Frequency = 261.625640869Hz, Step Size = 0.599187970161
note 60 off
note 64 on
Note: Frequency = 329.627685547Hz, Step Size = 0.754929602146
note 64 off
note 65 on
Note: Frequency = 349.228363037Hz, Step Size = 0.799820065498
note 65 off
note 67 on
Note: Frequency = 391.995574951Hz, Step Size = 0.897767663002
note 67 off
Exiting
```

Above The simple user interface. Notice how the step size in the wavetable varies with frequency

Performance issues

Python introduces a few performance issues compared to a synthesiser written in C or C++. Cython has been used in our implementation to try and mitigate these issues but it is nowhere near enough. As a rough comparison, our expert worked on a synthesiser project targeting 100 Mhz ARM processors that were programmed in C and could get around 30 notes of polyphony, compared to three notes in this implementation on a 900 Mhz ARM core.

The Code SIMPLE SYNTH

```
#!/usr/bin/python2

import pyaudio
import time
from array import *
from cpython cimport array as c_array
import wave
import threading
import tty, termios, sys
```

07

```
class MIDITable:
    # Generation code from
    # http://www.adambuckley.net/software/beep.c

    def __init__(self):
        self.notes = []
        self.fill_notes()

    def fill_notes(self):
        # Frequency of MIDI note 0 in Hz
        frequency = 8.175799

        # Ratio: 2 to the power 1/12
        ratio = 1.0594631

        for i in range(0, 128):
            self.notes.append(frequency)
            frequency = frequency * ratio

    def get_note(self, n):
        return self.notes[n]
```

The Code SIMPLE SYNTH

08

```
cdef class ADSR:
    cdef float attack, decay, sustain_amplitude
    cdef float release, multiplier
    cdef public char state
    cdef int samples_per_ms, samples_gone

    def __init__(self, sample_rate):
        self.attack = 1.0/100
        self.decay = 1.0/300
        self.sustain_amplitude = 0.7
        self.release = 1.0/50
        self.state = ‘A’
        self.multiplier = 0.0
        self.samples_per_ms = int(sample_rate / 1000)
        self.samples_gone = 0

    def next_val(self):
        self.samples_gone += 1
        if self.samples_gone > self.samples_per_ms:
            self.samples_gone = 0
        else:
            return self.multiplier
        if self.state == ‘A’:
            self.multiplier += self.attack
            if self.multiplier >= 1:
                self.state = ‘D’
        elif self.state == ‘D’:
            self.multiplier -= self.decay
            if self.multiplier <= self.sustain_amplitude:
                self.state = ‘S’
        elif self.state == ‘R’:
            self.multiplier -= self.release

        return self.multiplier
```

The Code SIMPLE SYNTH

09

```
cdef class Note:
    cdef int wavetable_len
    cdef float position, step_size
    cdef c_array.array wavetable
    cdef public float freq
    cdef public object adsr
    cdef public int off

    def __init__(self, wavetable, samplerate, freq):
        # Reference to the wavetable we're using
        self.wavetable = wavetable
        self.wavetable_len = len(wavetable)
        # Frequency in Hz
        self.freq = freq
        # The size we need to step though the wavetable
        # at each sample to get the desired frequency.
        self.step_size = self.wavetable_len * \
                                 (freq/float(samplerate))
        # Position in wavetable
        self.position = 0.0
        # ADSR instance
        self.adsr = ADSR(samplerate)
        # Is this note done with
        self.off = 0

    def __repr__(self):
        return ("Note: Frequency = {0}Hz, "
                "Step Size = {1}").format(self.freq,
                                          self.step_size)

    cpdef int next_sample(self):
        # Do the next sample
        cdef int pos_int, p1, p2, interpolated
```

The Code SIMPLE SYNTH

09

```
        cdef int out_sample = 0
        cdef float pos_dec
        cdef float adsr

        adsr = self.adsr.next_val()
        # Need to turn the note off
        # synth will remove on next sample
        if adsr < 0:
            self.off = 1
            return out_sample

        pos_int = int(self.position)
        pos_dec = self.position - pos_int

        # Do linear interpolation
        p1 = self.wavetable[pos_int]
        p2 = 0

        # Wrap around if the first position is at the
        # end of the table
        if pos_int + 1 == self.wavetable_len:
            p2 = self.wavetable[0]
        else:
            p2 = self.wavetable[pos_int+1]

        # Inerpolate between p1 and p2
        interpolated = int(p1 + ((p2 - p1) * pos_dec))
        out_sample += int(interpolated * adsr)

        # Increment step size and wrap around if we've
        # gone over the end of the table
        self.position += self.step_size
        if self.position >= self.wavetable_len:
```

The Code SIMPLE SYNTH

09

```
                self.position -= self.wavetable_len

            return out_sample

class Synth:
    BUFSIZE = 1024
    SAMPLERATE = 44100

    def __init__(self):
        self.audio = pyaudio.PyAudio()

        # Create output buffers
        self.buf_a = array('h', [0] * Synth.BUFSIZE)
        self.buf_b = array('h', [0] * Synth.BUFSIZE)
        # Oldbuf and curbuf are references to buf_a or
        # buf_b, not copies. We're filling newbuf
        # while playbuf is playing
        self.playbuf = self.buf_b
        self.newbuf = self.buf_a

        self.load_wavetable()
        self.notes = []
        self.notes_on = []

        # The synth loop will run in a separate thread.
        # We will use this condition to notify it when
        # we need more samples
        self.more_samples = threading.Event()
        self.exit = threading.Event()

        # MIDI table of notes -> frequencies
        self.midi_table = MIDITable()
```

```
    def stop(self):
        print "Exiting"
        self.exit.set()
        self.stream.stop_stream()
        self.stream.close()

    def stream_init(self):
        self.stream = self.audio.open(
                    format = pyaudio.paInt16,
                    channels = 1,
                    rate = Synth.SAMPLERATE,
                    output = True,
                    frames_per_buffer = Synth.BUFSIZE,
                    stream_callback = self.callback)

    def load_wavetable(self):
        # Load wavetable and assert it is the
        # correct format
        fh = wave.open('square.wav', 'r')
        assert fh.getnchannels() == 1
        assert fh.getframerate() == Synth.SAMPLERATE
        assert fh.getsampwidth() == 2 # aka 16 bit

        # Read the wavedata as a byte string. Then
        # need to convert this into a sample array we
        # can access with indexes
        data = fh.readframes(fh.getnframes())
        # h is a signed short aka int16_t
        self.wavetable = array('h')
        self.wavetable.fromstring(data)

    def swap_buffers(self):
        tmp = self.playbuf
        self.playbuf = self.newbuf
```

The Code SIMPLE SYNTH

```
        self.newbuf = tmp
        # Setting the condition makes the synth loop
        # generate more samples
        self.more_samples.set()

    def callback(self, in_data, frame_count,
                 time_info, status):
        # Audio card needs more samples so swap the
        # buffers so we generate more samples and play
        # back the play buffer we've just been filling
        self.swap_buffers()
        return (self.playbuf.tostring(),
                pyaudio.paContinue)

    def do_sample(self, int i):
        cdef int out_sample = 0
        # Go through each note and let it add to the
        # overall sample
        for note in self.notes:
            if note.off:
                self.notes.remove(note)
            else:
                out_sample += note.next_sample() >> 3

        self.newbuf[i] = out_sample

    def synth_loop(self):
        cdef int i

        while self.exit.is_set() == False:
            # For each sample we need to generate
            for i in range(0, Synth.BUFSIZE):
                self.do_sample(i)
```

The Code SIMPLE SYNTH

12

```
                # Wait to be notified to create more
                # samples
                self.more_samples.clear()
                self.more_samples.wait()

    def start(self):
        self.stream_init()
        # Start synth loop thread
        t = threading.Thread(target=self.synth_loop)
        t.start()

    def freq_on(self, float freq):
        n = Note(self.wavetable, Synth.SAMPLERATE,
                 freq)
        print n
        self.notes.append(n)

    def freq_off(self, float freq):
        # Set the ADSR state to release
        for n in self.notes:
            if n.freq == freq:
                n.adsr.state = ord('R')

    def note_on(self, n):
        self.freq_on(self.midi_table.get_note(n))
        self.notes_on.append(n)

    def note_off(self, n):
        self.freq_off(self.midi_table.get_note(n))
        self.notes_on.remove(n)

    def toggle_note(self, n):
```

12

```
        if n in self.notes_on:
            print "note {0} off".format(n)
            self.note_off(n)
        else:
            print "note {0} on".format(n)
            self.note_on(n)
```

13

```
class KBInput:
    def __init__(self, synth):
        self.synth = synth

        self.keymap = {'a' : 60, 'w' : 61, 's' : 62,
                       'e' : 63, 'd' : 64, 'f' : 65,
                       't' : 66, 'g' : 67, 'y' : 68,
                       'h' : 69, 'u' : 70, 'j' : 71,
                       'k': 72}
        self.notes_on = []

    @staticmethod
    def getch():
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(fd)
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN,
                              old_settings)
        return ch

    def loop(self):
        while True:
            c = self.getch()
```

13

```
        if c == 'q':
            self.synth.stop()
            return

        if c in self.keymap:
            n = self.keymap[c]
            self.synth.toggle_note(n)

if __name__ == "__main__":
    s = Synth()
    s.start()
    kb = KBInput(s)
    kb.loop()
```

RasPiViv

Nate Bensing tells us how he built an environmental control system to keep seven poison dart frogs cosy

So, what do you keep in the vivarium?

Right now I have seven poison dart frogs – they're frogs from South America that, in the wild, excrete poison alkaloids, but in captivity, because their diet is just fruit flies, they can't produce any poison. They're something I've been interested in for quite a long time. I think I saw them first when I was in grade school at a trip to the Denver zoo, and I just thought they were the coolest animals I'd ever seen in my life. I've wanted to keep them since then but the opportunity never came up – they're kinda rare – until I found a breeder on Craigslist who has an incredible collection and he breeds them to support his hobby. So right now I have a total of seven: two blue poison dart frogs, which I think could be a breeding pair, and I recently obtained five yellow-banded poison dart frogs.

What kind of requirements do you have for the vivarium, then?

The temperature is really important, which a lot of people have trouble with because your house temperature is going to be about the same as an enclosed box, give or take, as lighting can heat things up. But you have to maintain specific temperatures between about 75 and 85 degrees, and the humidity is even more important because the frogs use the humidity to thermoregulate, kinda like how we sweat.

So basically, what I needed was a way to monitor and regulate the humidity. I looked around online for a couple of days trying to come up with a solution – a lot of people use little timers, and there are a couple of systems that are made for this but they don't do very much.

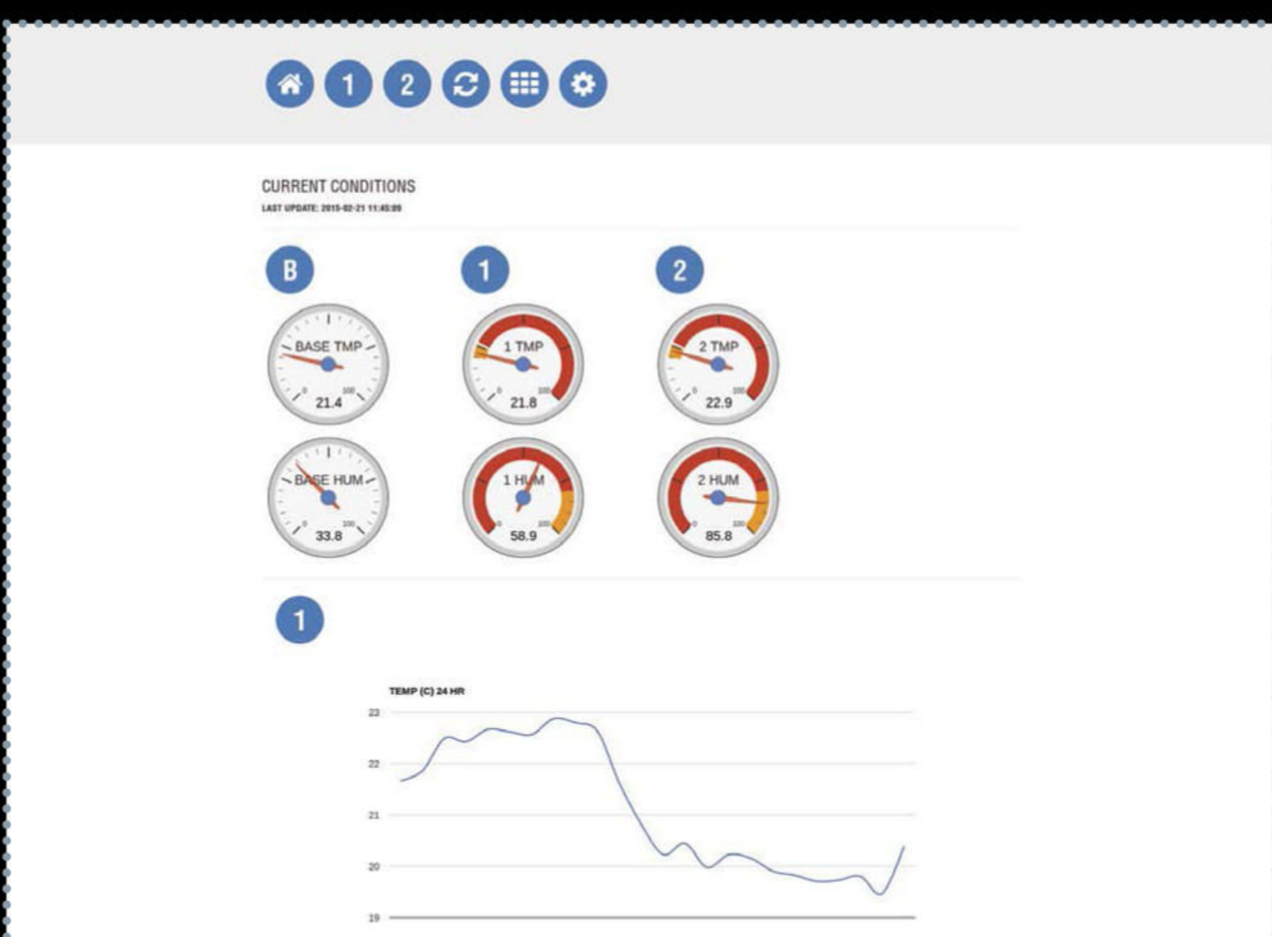
What hardware did you use to make your own solution?

Well, the Raspberry Pi is set up as a LAMP server. I started

Nate Bensing is a marketing consultant, photographer and graphic designer for Wheaton Design in Northern Colorado. He enjoys learning about electronics, creating automation projects, and contributing as much as he can to the Pi and Linux communities that made this possible.

playing around with the DHT22 temperature-humidity sensor – originally I just wanted to monitor that, the temperature and the humidity, but then I got into using relays to control the lighting and it just progressed further and further. I was just making this for myself and I posted on a forum. Then a lot of people seemed really interested, so I said I'd clean things up and throw together a guide (see **www.raspiviv.com**).

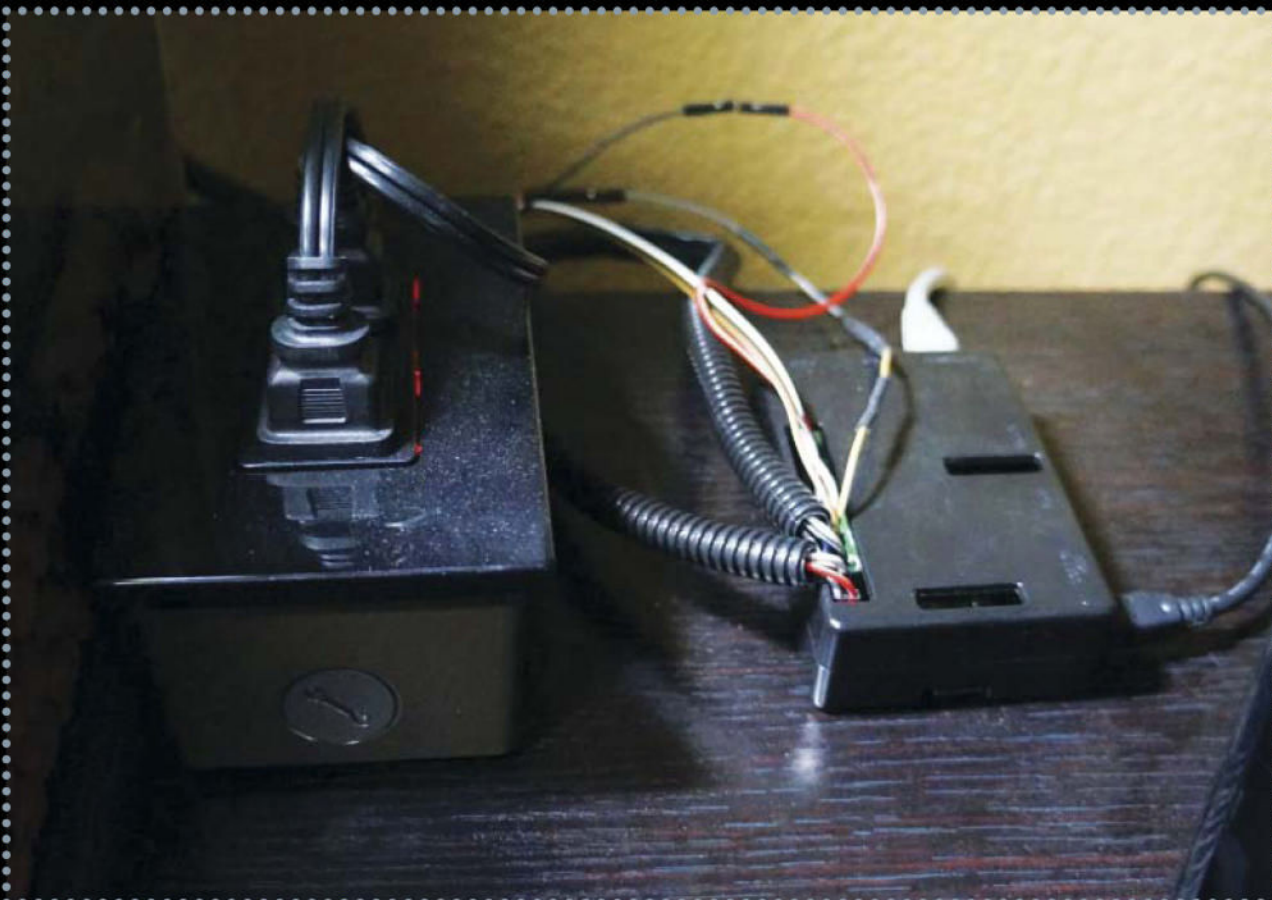
So the temperature and humidity sensors are read every 60 seconds and logged into a database. Using WiringPi and Adafruit's DHT22 sensor (and the driver for it, which is brilliant and works really well), lighting is controlled by regular relays or relay modules, and the fan is just a basic transistor switch. The main idea behind the fan is that if the atmosphere is too saturated with water then the frogs can't thermoregulate. So the database is read every five minutes, and when it reaches 95% humidity it then kicks on the fan to blow in some fresh air.



If you like

Fancy building your own Vivarium controller? Check out the excellent step-by-step guide on Nate's website that takes you from NOOBS to the humidity regulation cron job (**http://bit.ly/1HTKyeX**).

Left The RasPiViv web interface shows you the temperature and humidity readings over time



Do you SSH in to control all this or do you have a web interface for it?

Yeah, there's a whole web interface where you can check out the current readings. Check out the Demo section on my website and the first thing that pops up is my blue poison dart frog vivarium. It gives you the current temperature and humidity, and then also the readings for the last hour. If you click on the icon that looks like a grid of buttons (manual controls), you can manually control your lighting, any misting system or fans you have – essentially, for any component that you want to control, it's just a matter of getting a relay module and wiring it up.

Are you planning to upgrade the RasPiViv software at any point?

Yeah, I am hoping to. I started work on this in my downtime and got insanely busy, so unfortunately I haven't been able to do a lot with it. I'm hoping to get some RF power outlets soon so that, rather than wiring up a little power outlet

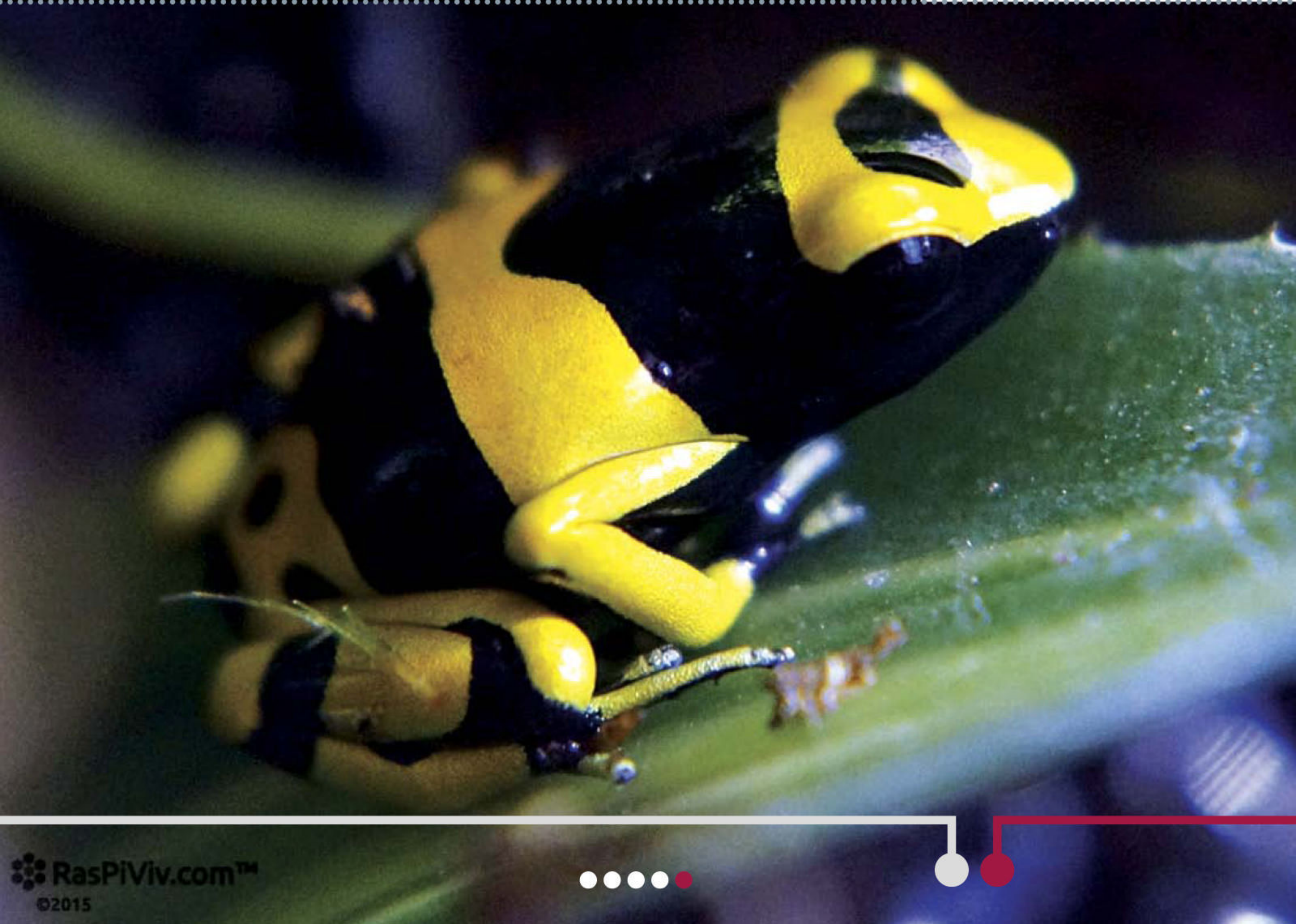
Further reading

Looking for more inspiration for sensor-driven projects? There are some great ones featured on The Raspberry Pi Foundation blog, like the Feeder Tweeter and the PiPlanter: **http://bit.ly/1Ak37mu**.

box, you can just use these prebuilt outlets that you plug into your wall and they give you a little remote control. I am hoping to implement some wireless stuff and I'd definitely like to make it more user friendly, rather than people manually adding cron jobs to things. I'd like them to be able to do it through the browser interface, stuff like that.

What you don't see in the demo is that there's also a login system – you can make an account and keep it secure – so I want to give people that and I'll probably run a tutorial for setting it up. I've been playing around with the Raspberry Pi camera module and hoping to include it, so now we have a Raspberry Pi 2 that is a lot more capable, it could potentially pull off some kind of live camera that you can watch as well as all the other stuff that it does.

Below Nate uses his setup to keep frogs, but you could adapt the temperature monitor for a reptile or even a fish tank!

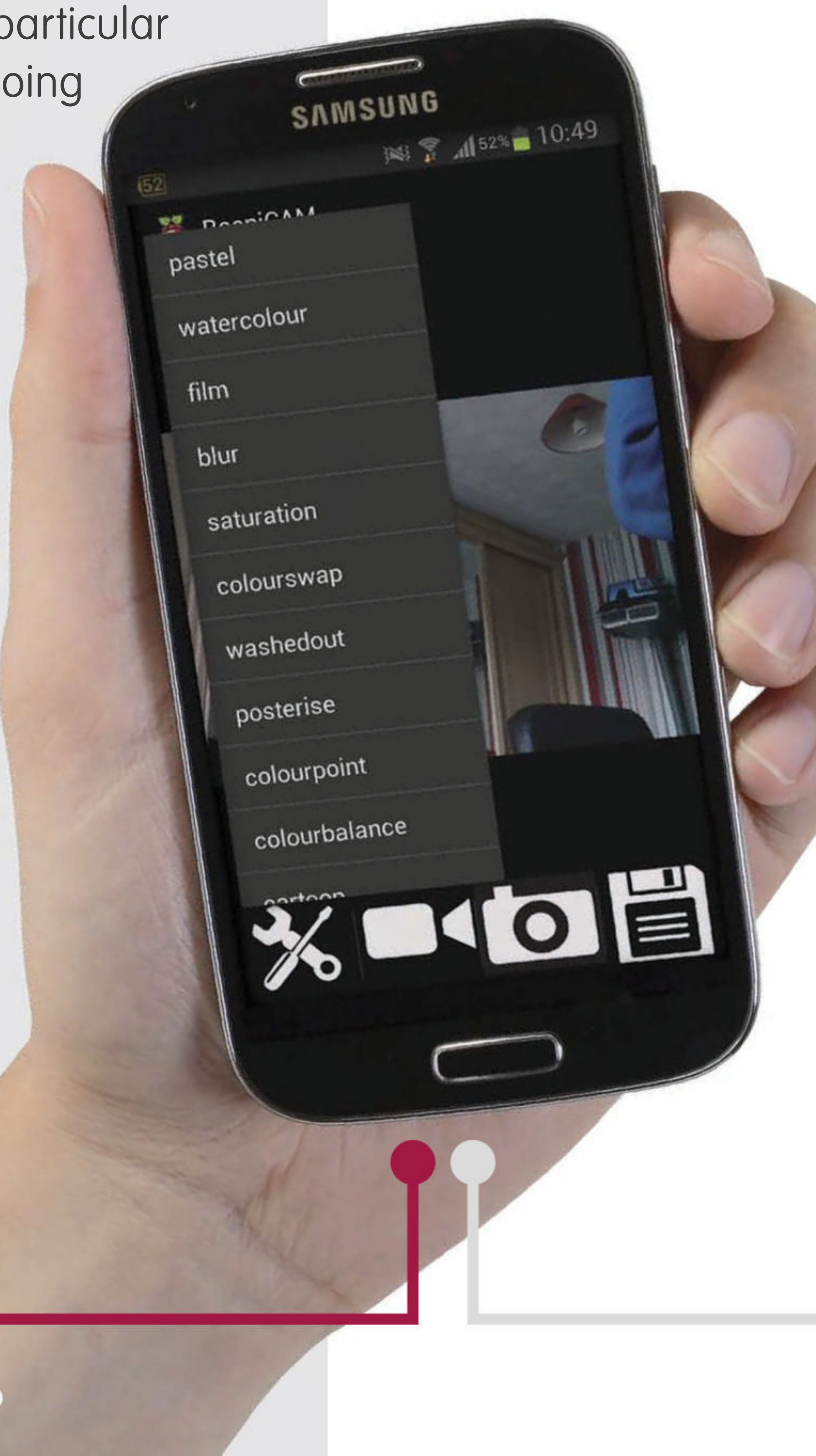


Top Android apps for your Pi

Download the most useful tools for your Pi onto the device you carry the most

Mostly, our tutorials are about completing a specific project and reaching a particular goal. However, this time we're doing something a bit different. We are showing you some Android apps that you can use along with your Ras Pi. These apps aren't tied to particular projects – you can use them whenever and as often as you like – but we think they can add something to your whole experience with the Pi.

Some of the apps in our list are Pi-specific, while others are more general but have a Pi relevance. Chances are you might already know or use one or two, but we hope that you can discover something new from the selection on offer. If you have an Android phone or tablet and have not explored the range of apps available for your Raspberry Pi, you might be missing out on some cool and very useful options.



Below RasPiCam is great for remote video viewing

RaspiCam Remote

If you are into apps that use the Pi camera then here's a neat free app that lets you access it remotely. You can save images to your Android device so you can access them later and it will run a video mode based on single-frame capture.

AndFTP (your FTP client)

Sometimes you might need to transfer files onto your Pi, and for this you may appreciate AndFTP. It can serve more devices than the Pi and could be useful if your Pi is a web server. The free version is competent, while the Pro version at £3.89 adds in features like folder sync.

Raspi-R

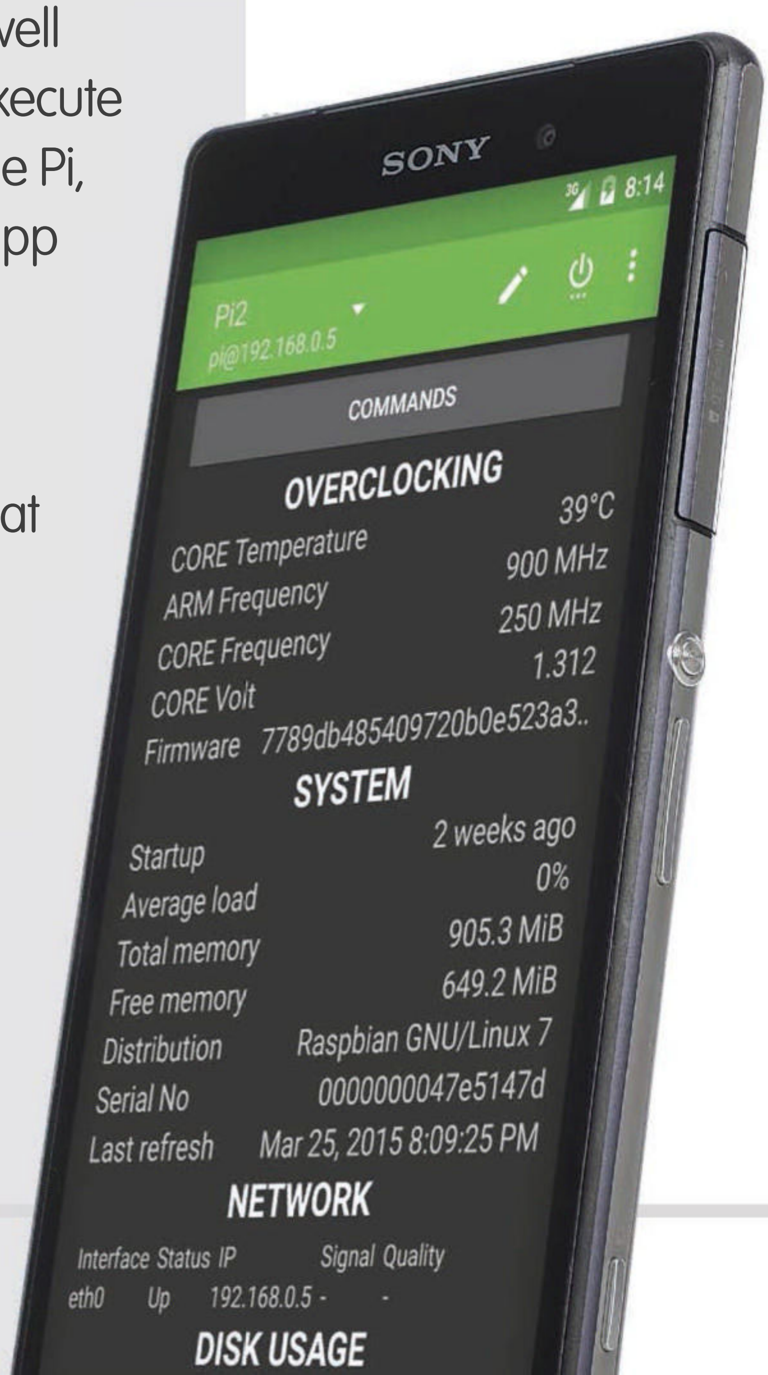
This is a well-featured remote control app that can accommodate more than one Pi. As well as monitor the status of your Pi, you can execute Linux commands, remotely control and reboot the Pi, and handle Python3 scripts. A free app with in-app purchases available.

Fing – Network Tools

Fing is a general network scanning app that identifies everything on your network by name and delivers both their IP and MAC addresses. You can also send a ping if you feel the need. It really is useful because we all forget our Pi's IP address from time to time.

RasPi Check

When you want to check info like the free memory of your Pi or its network status, processes and such, set up SSH login and use the free RasPi Check on your Android tablet or phone. The app can also restart and stop the Pi, and send custom commands, too.



Raspberry Pi Remote

If you want to be able to control services from afar then the £0.66 Raspberry Pi Remote could be the app for you. Install it and you can control things like the camera module at long range, as long as your Pi is Internet-enabled.

JuiceSSH – SSH client

If you are using SSH to control your Pi from another device, why not add your Android tablet or phone into the mix? There are a number of terminal clients for Android and for SSH with multiple devices – we quite like the free JuiceSSH app because it has a nice, clear interface.

RPiREF

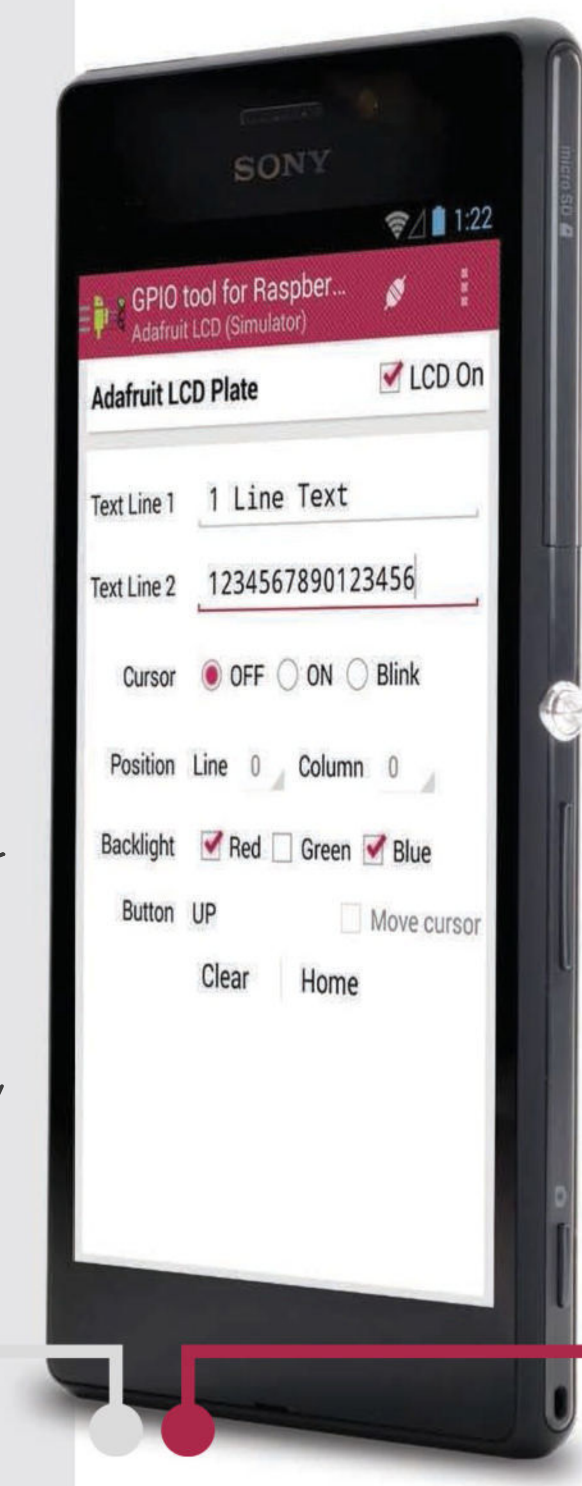
This app is probably better suited to a larger-screened tablet than a small handset because it is a reference guide to the GPIO pinout, covering the A, B and B+, (the A+, Pi 2 and Pi 3 layouts are identical to the B+). The info is really clearly set out, and it's free.

VNC Viewer

For even more direct remote control of your Ras Pi, we recommend checking out VNC Viewer. This is an excellent VNC client that enables you to view and control your Pi's desktop GUI if you've got a VNC server running on it. RealVNC, makers of the VNC Viewer app, have an open source version of their VNC software available to download on their website, and you can also get hold of a free license for the main VNC software, which is only for personal use and does not include encryption. We like using the tightvnc server.

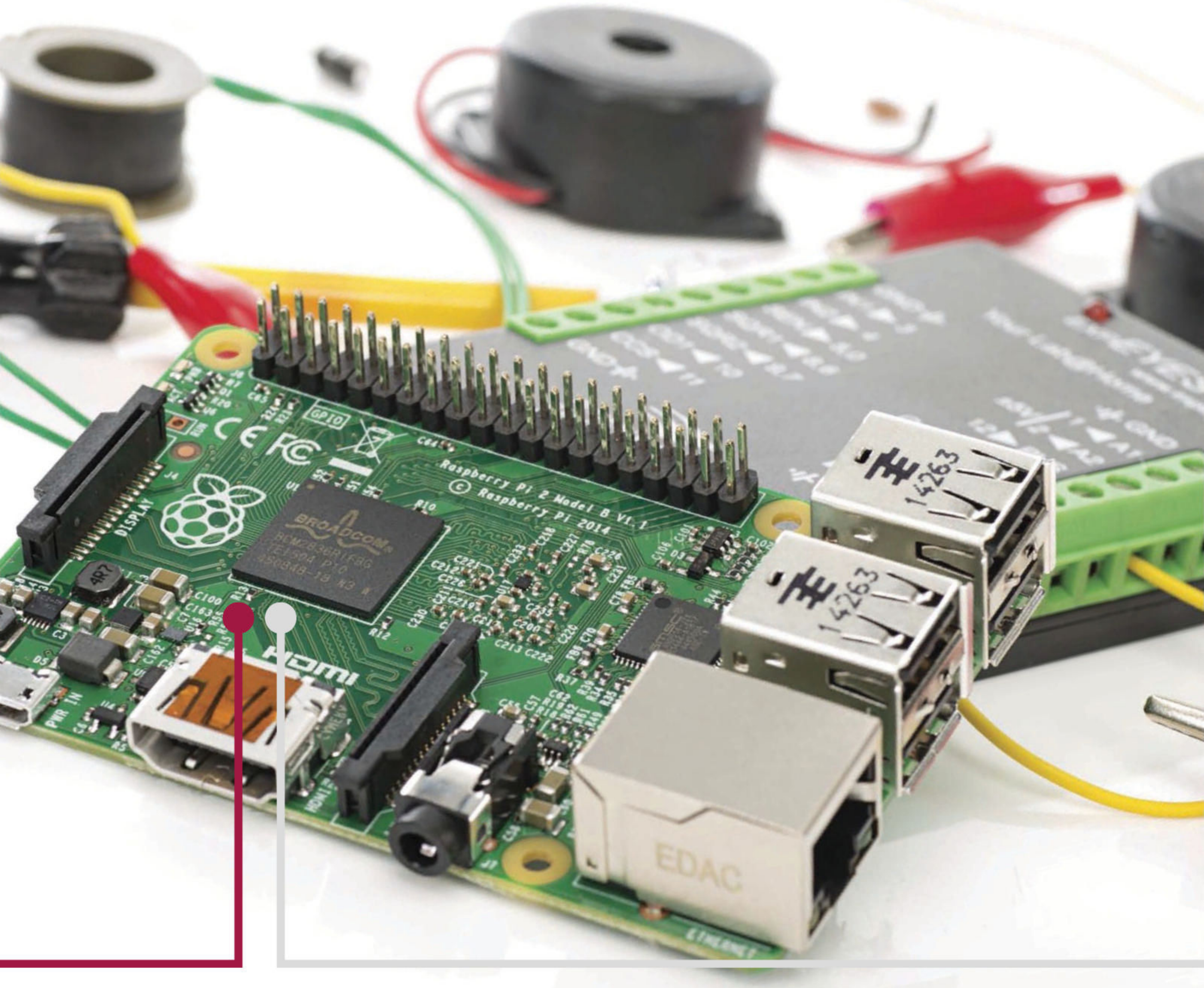
GPIO Tool For Raspberry Pi

Some people have a list of Pi project ideas, but lots of us are always looking for inspiration. With this app, you can do a few simulations. It is a mixed bag but worth a look. It's free, with in-app purchases.



Run science experiments on the ExpEYES kit

ExpEYES is an affordable digital oscilloscope with a signal generator, and other features – perfect for electronics



ExpEYES is a relatively unheard of but very impressive hardware and software platform for science and electronics experimentation, as well as a useful electronic probing tool for makers and professionals alike. It is also open source on both the hardware and software sides, which makes it affordable and versatile.

ExpEYES is billed as a science and experimentation kit but really it is much more than that – it is a fully-functioning four-channel digital oscilloscope with an impressive array of features. ExpEYES ships with a wealth of online documentation in a variety of formats (graphics, user guides, web content), including upwards of 50 suggested experiments, and the kit itself contains all of the hardware required to play around with the interesting science of electronics contained within the guide material.

The aim is to enable the learning of what can be complex concepts of electronics in an easy and affordable way, without getting bogged down in the arcane details. Paired with our favourite little single-board computer, the Raspberry Pi, you have an extremely powerful and affordable device.

THE PROJECT ESSENTIALS

Raspberry Pi
ExpEYES kit
http://bit.ly/1AR15dz

"The aim is to enable the learning of what can be complex concepts of electronics in an easy and affordable way"

01 Get the parts

ExpEYES is available to purchase from a variety of online vendors, including CPC (**http://cpc.farnell.com**), for around £50. It is possible to get the kits slightly cheaper from India (see **http://expeyes.in/hardware.html** for other vendors worldwide), however it's likely to end up costing more due to higher shipping rates as well as potential import fees and duties.

02 Open it up

The ExpEYES kit contains everything you need to get underway, with over 50 documented experiments from the ExpEYES website. The only other item that may come in handy is a breadboard. You will also need a Raspberry Pi, or another computer with a USB port, in order to run the digital oscilloscope software and connect to ExpEYES.

03 What's inside?

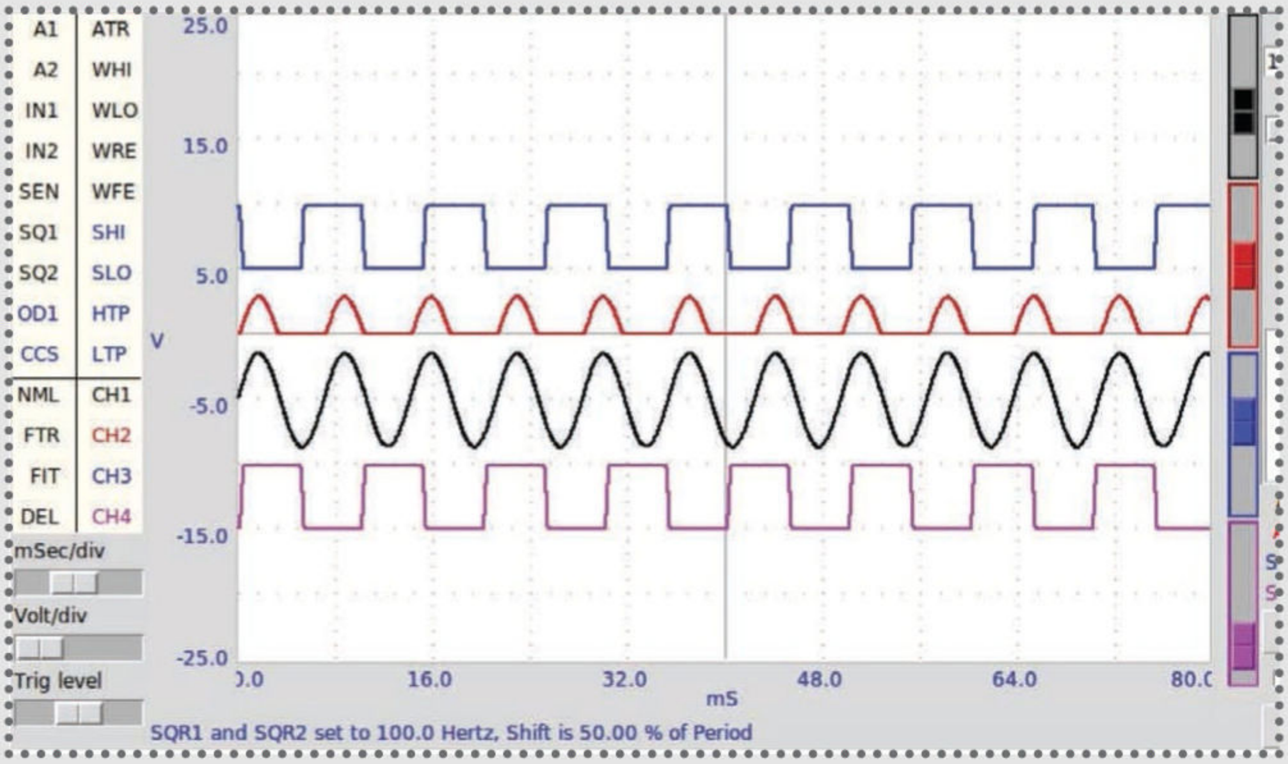
As you may have guessed, the ExpEYES kit includes the main ExpEYES USB digital oscilloscope, but it also contains a wide range of other hardware including a DC motor, magnets, LEDs, coils, piezoelectric discs, wiring, a small screwdriver for opening the screw terminals and more. You also get a live CD which contains all the ExpEYES software and documentation ready to go on a bootable disc.

04 What can it do?

The chip at the heart of ExpEYES is an AVR ATmega16 MCU (microcontroller unit), running at 8 MHz coupled to a USB interface IC (FT232RL). These are low-cost but provide good value for money. As we have already mentioned, ExpEYES is therefore capable of acting as a four-channel oscilloscope but also has a built-in signal generator, 12-bit analogue resolution, microsecond timing resolution and a 250 kHz sampling frequency. At this price point, that's an impressive set of features and certainly accurate enough for anything that is not mission critical (like learning, hobby projects, quick readings and so on).

Supported platforms

The ExpEYES software is mainly written in Python. This means that the core software to run your ExpEYES device is quite platform-agnostic – if the device can run a Python interpreter and has a Python module enabling it to access the serial port, then it will work with ExpEYES. If you visit the ExpEYES website, there is a page that explains how to install the software on Linux and Windows – **http://www.expeyes.in/software.html**.



05 Using the live CD

Perhaps the easiest way to get up and running with ExpEYES (if you have a computer with a CD drive) is to use the live CD which is included in the ExpEYES kit. Making sure that you are booting into the live CD from your BIOS boot menu, you should then be greeted with a Linux-based desktop. Plug in your ExpEYES by USB and you can open the software from the menu by going to Applications>Science>ExpEYES-Junior. Alternatively, you can run it from a terminal window using:

```
sudo python /usr/share/expeyes/eyes-junior/
croplus.py
```

06 Update your Raspberry Pi

As with almost every project you undertake on the Raspberry Pi, it pays dividends to make sure that your operating system is updated to the latest stable version, as this can save you a lot of hassle further down the line. To do this, open an LXTerminal session and then type **sudo apt-get update**, followed by **sudo apt-get upgrade –y**, and then wait patiently for the upgrade process to complete.

07 Install the software

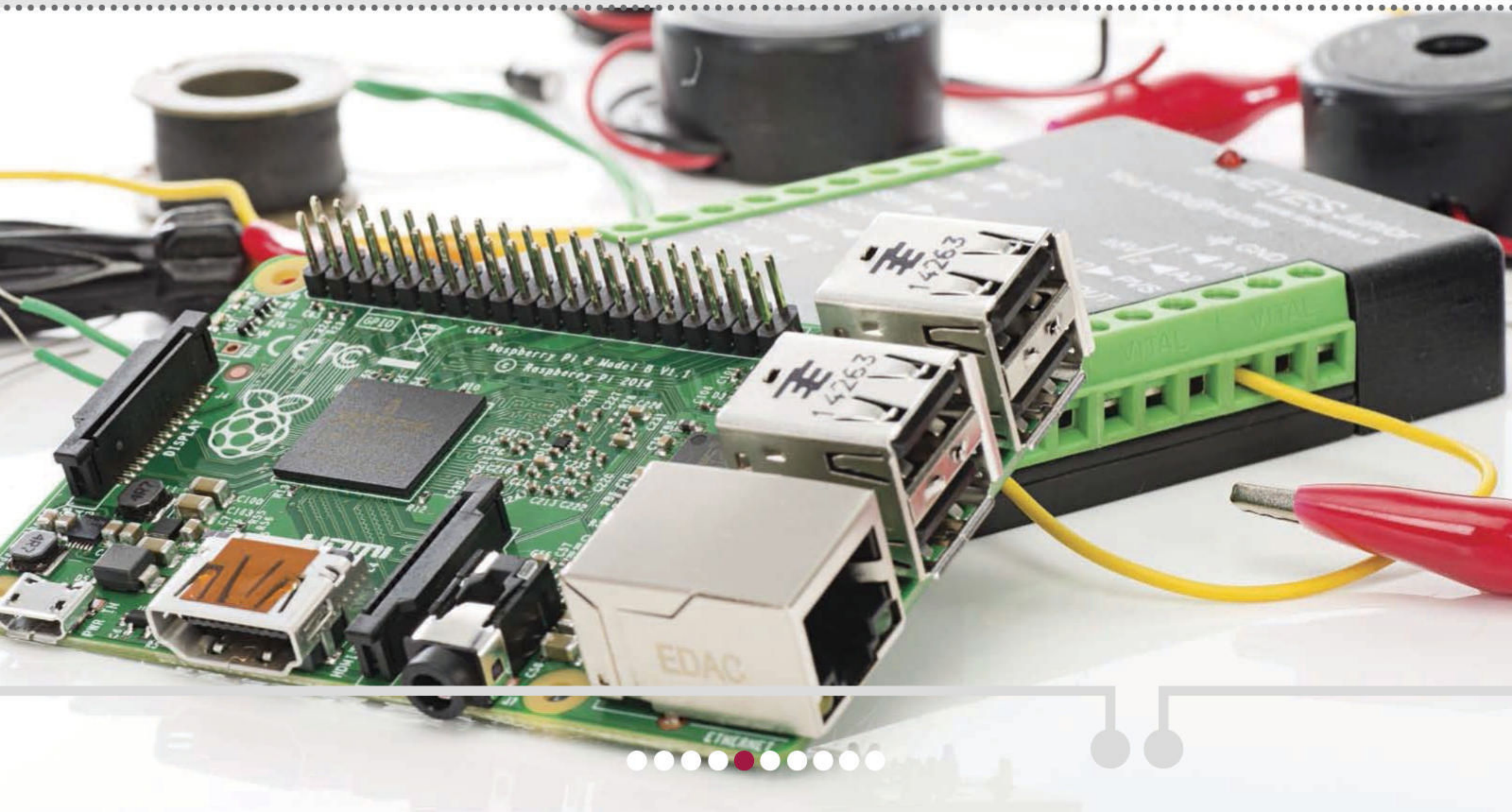
Due to efforts of community member Georges Khaznadar, there are DEB packages available for the ExpEYES software that should work perfectly on Debian, Ubuntu, Linux Mint and, of course, Raspbian. These are also included in the official Raspbian repositories, so all you need to do to install the ExpEYES software is to open an LXTerminal session on the Raspberry Pi and then run the following commands:

```
sudo apt-get update
sudo apt-get install expeyes
```

08 Install dependencies

ExpEYES has a number of dependencies that are required for it to run under Linux, as well as a number of other recommended libraries. During the installation undertaken in Step 7, the dependencies should be installed by default. However, to avoid any problems later, you can run the following command in order to make sure that they are all installed:

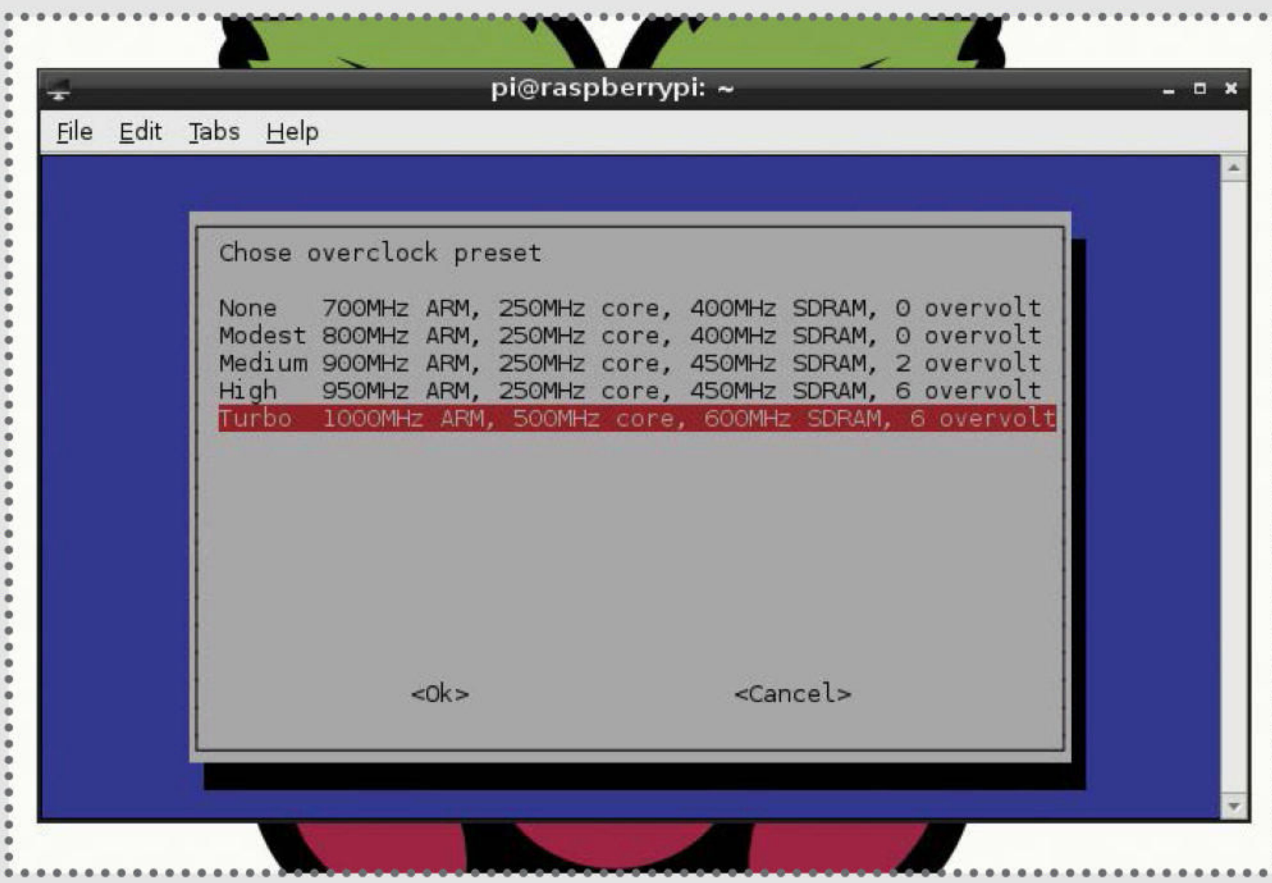
Above The ExpEYES hardware has a female header, making it simple to plug in jumper wires



```
sudo apt-get install python python-expeyes
python-imaging-tk python-tk grace tix python-
numpy python-scipy python-pygrace
```

09 Overclock your Raspberry Pi (optional)

The ExpEYES software will run fine on a Raspberry Pi with default settings, however it can be slow to respond if you are using a Model A, B or B+. We recommend using a Model 2B, but if you don't have one, overclocking your Pi would be advisable (you can overclock your 2B as well if you want it to run a bit faster). Open an LXTerminal session and type **sudo raspi-config**. In the menu, select the option '7 Overclock'. Click OK on the following screen and then select Turbo. Click OK and you should see some code run. Once this completes, press OK again and then you are brought back to the main raspi-config window. Select Finish in the bottom right and Yes to reboot your Raspberry Pi.



10 Overclocking continued

Overclocking can sometimes cause instability on your Raspberry Pi or an inability to boot at all. If this happens you can press and hold the Shift key on your keyboard once you reach the above splash screen to boot into recovery mode. You can then redo Step 7 at a lower overclock setting and repeat until you find the highest stable setting.

11 Resistance of the human body

An interesting experiment for your first time using an oscilloscope it to measure the resistance of the human body over time. This is easy to accomplish with just three bits of wire and a resistor (200 kOhm). On the ExpEYES, connect a wire between A1 and PVS, connect the resistor between A2 and ground, and connect an open-ended wire out of both PVS and A2. Plug in your ExpEYES and open the control panel, then drag A1 to CH1 and A2 to CH2, and set PVS to 4 volts. You can then pick up one of the open-ended wires in each hand and watch the response on the ExpEYES control panel.

12 Run the maths

From the output plot, you should find that the input on CH1 is coming out at 3.999 volts (which is great because we set it to be 4!). The voltage on A2 (CH2) is showing as 0.9 volts for us, which implies that the voltage across the unknown resistor value (your body) is 4 – 0.9 = 3.1 volts. Using Ohm's law (V=IR), we can then calculate the current (I) across the known resistor value: voltage ÷ resistance = 0.9 ÷ 200,000 = 0.0000045 amps = 4.5 uA (micro amps). Using this value we can then calculate the resistance of the

ExpEYES & PHOENIX

ExpEYES was developed by Ajith Kumar and his team as part of the PHOENIX (Physics with Homemade Equipment and Innovative Experiments) project, which was started in 2005 as a part of the outreach program of the Inter-University Accelerator Centre (IUAC) in New Delhi, India. Its objectives are developing affordable laboratory equipment and training teachers to use it in their lesson plans.

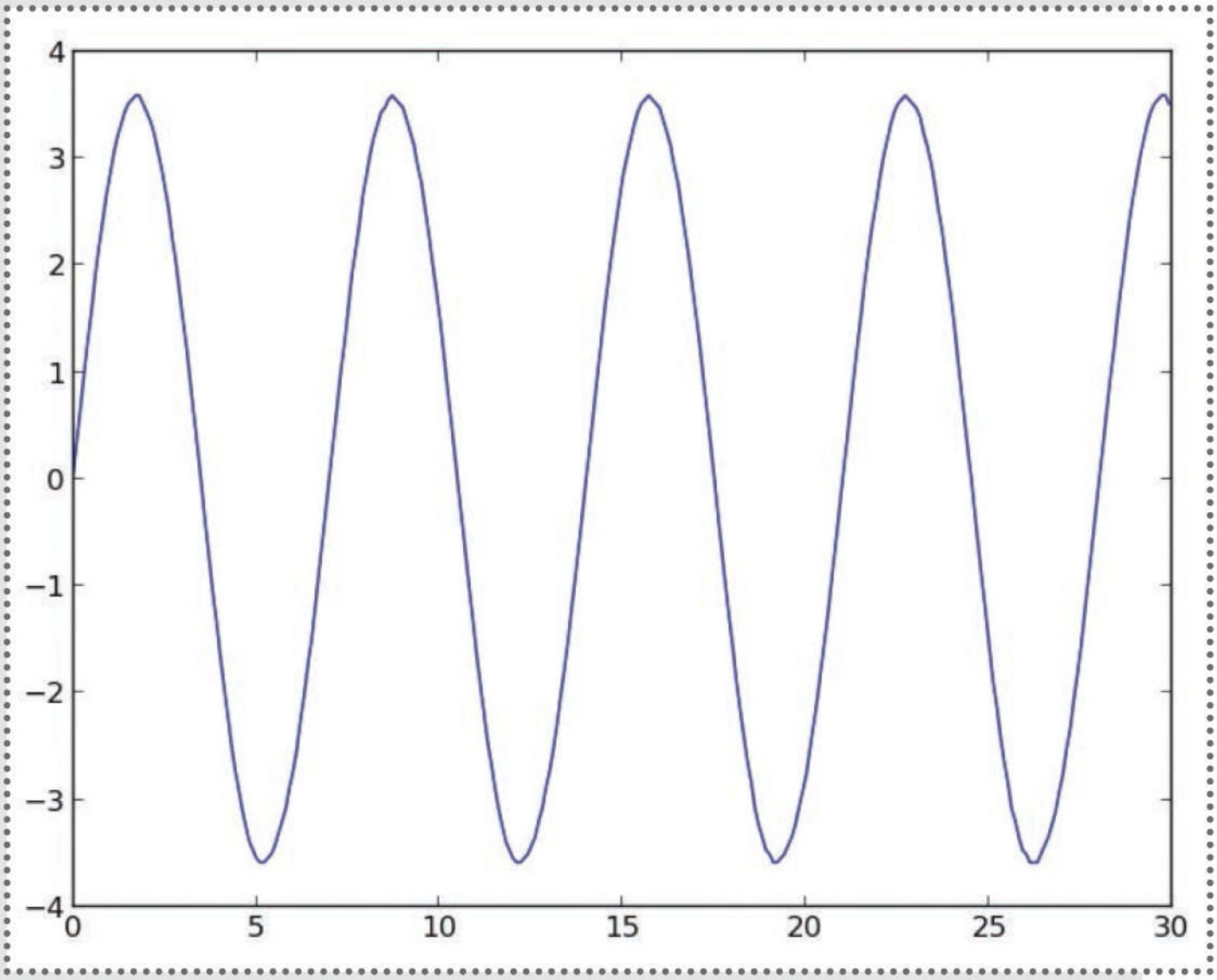
Above Each of the inputs is clearly labelled, and there are plenty of guides included in the box

body using the same Ohm's law equation in reverse: voltage ÷ current = 3.1 ÷ 0.0000045 = 688889 ohms = 689 kΩ. This is a surprisingly high value, however the resistance of the human body depends hugely on how dry your skin is and a large number of other factors (body resistance is usually in the range of 1,000 to 100,000 ohms).

13 Use the Python library

The ExpEYES team have built a custom Python library for the device. This is slightly harder to use than the GUI and not as pretty, but it enables a lot more versatility as well as the capability to use ExpEYES functionality within your Python scripts. If you have followed the installation instructions above, all you need to do is import the Python module and then initialise a connection to the ExpEYES using:

```
import expeyes.eyesj
p=expeyes.eyesj.open()
```



14 The Python library (continued)

Now we will plot a sine wave using the ExpEYES and PyLab libraries. On the device, connect OD1 to IN1 and SINE to A1 with some wire. Run the following code and you should see that a sine wave has been plotted.

```
import expeyes.eyesj
from pylab import *

p=expeyes.eyesj.open()
p.set_state(10,1)
print p.set_voltage(2.5)
ion()     # set pylab interactive mode
t,v = p.capture (1,300,100)
(plot t,v)
```

"The custom Python library is slightly harder to use than the GUI and not as pretty, but it enables a lot more versatility"

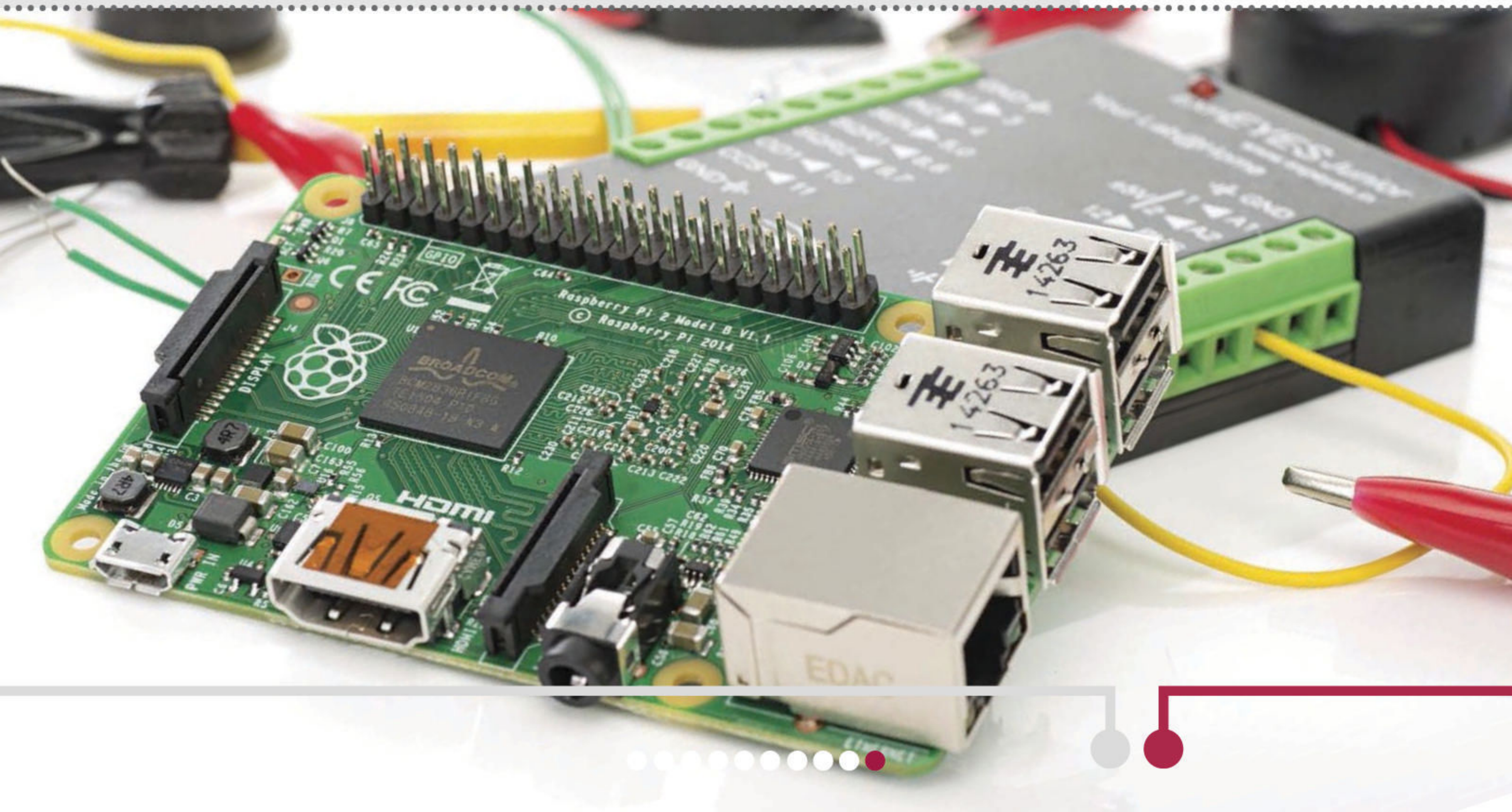
15 Further experiments

This tutorial has shown you just a single example of the documented ExpEYES experiments available at **http://expeyes.in**. There is a wide variety of different techniques and phenomena explored in those experiments, so it is highly recommended to get your hands on an ExpEYES kit and work through them. Running through those examples as a beginner will give you a much deeper understanding of electronics.

16 The verdict

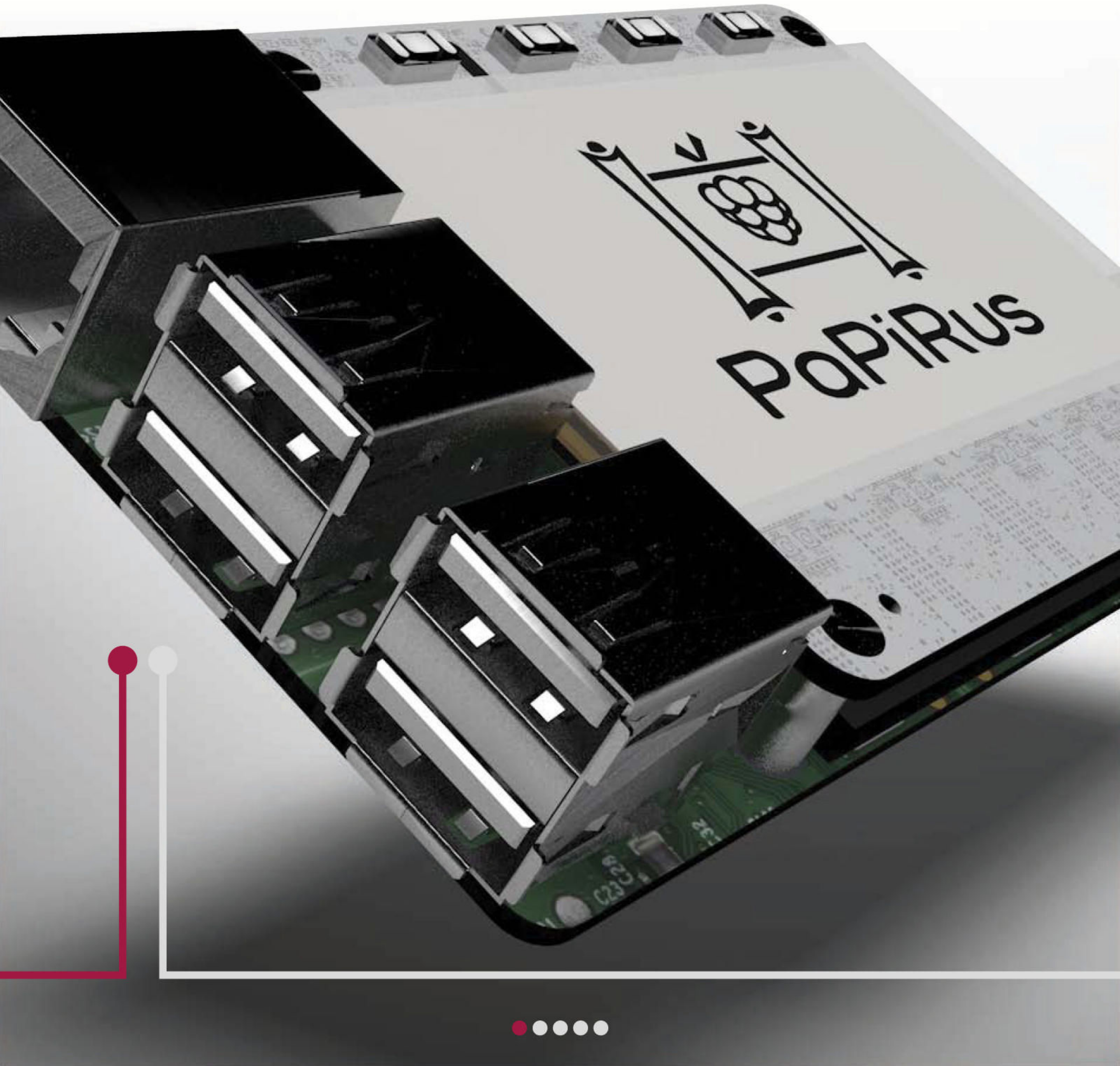
A digital storage oscilloscope (plus extras) is a useful tool in any engineer or hacker's toolbox, as it enables you to get insights into your projects that aren't possible with just visual checks or using a multimeter. Whilst no £50 oscilloscope will compare to expensive professional units, this is a great entry-level product as well as a versatile, portable USB device with multiplatform support for when you just can't be lugging around a 10 kg, £1000+ scope.

Below The project documentation can all be found on GitHub: **https://github.com/expeyes/expeyes-doc**



FAQ What is PaPiRus?

Need a low-cost, low-power display for your Pi project? Check out the amazing PaPiRus HAT!

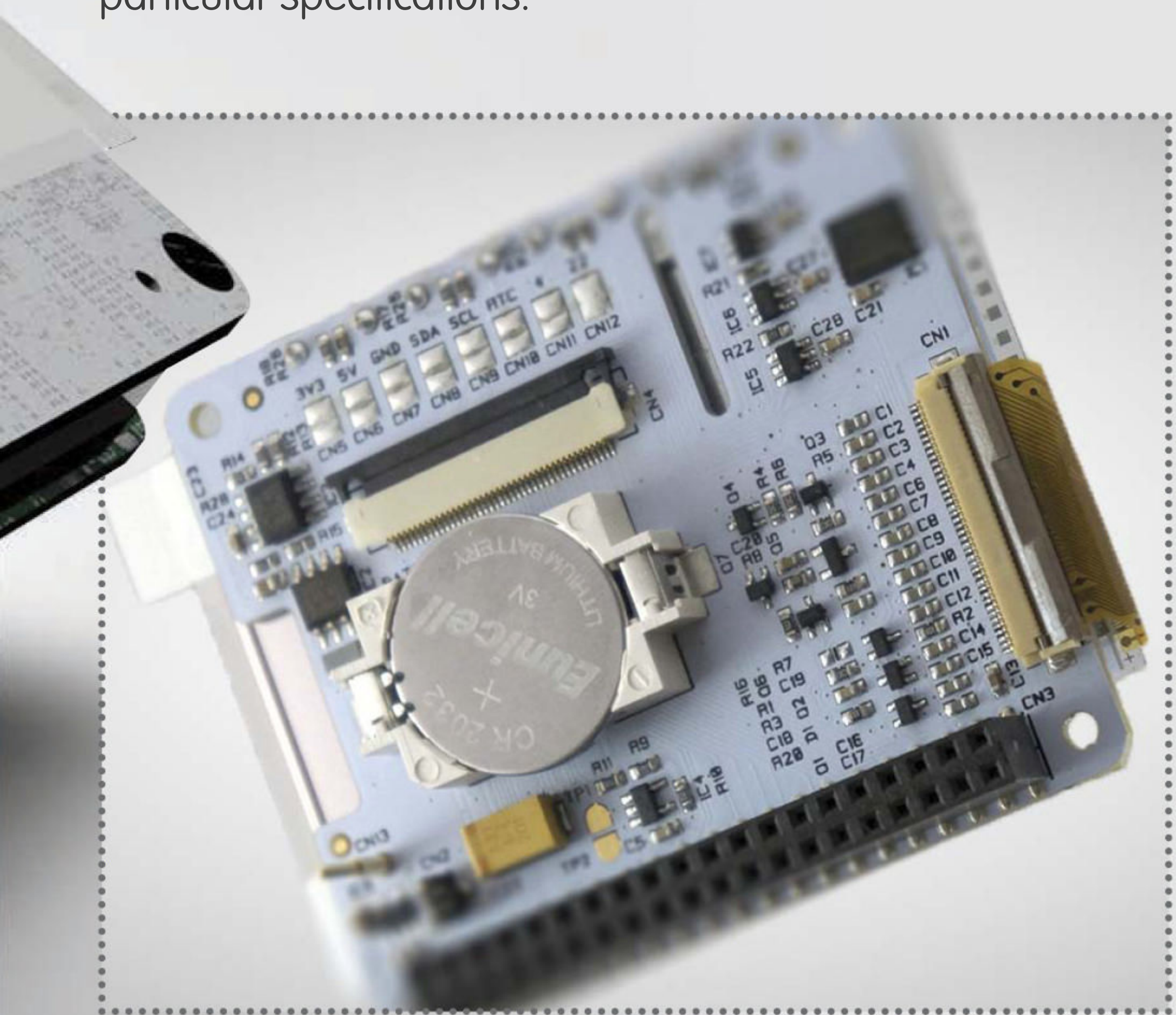


I'm guessing this is not about vintage papercraft, then?
Not this time, although the PaPiRus is quite paper-like in many ways.

Well, now I'm confused. So what exactly is the PaPiRus?
Well, the PaPiRus is a HAT module for the Raspberry Pi that adds an ePaper display to it for you to use in your projects.

Remind me again – what exactly is a HAT module?
The Raspberry Pi Foundation released something called the HAT specification, which is a set of guidelines for producing Raspberry Pi add-ons that sit nice and neatly on top of the Pi itself – Hardware Attached on Top – and which also makes things much easier for their end users. So the PaPiRus is an add-on board that fits in with the Foundation's particular specifications.

"Well, the PaPiRus is a HAT module for the Raspberry Pi that adds an ePaper display to it for you to use in your projects"



Left The PaPiRus HAT runs a real-time clock off a watch battery, which is very nice

Next one: what's ePaper – the stuff they use in the Kindle?

Pretty much, yes. ePaper has seen use most famously in the Kindles and the Pebble watch, and the key things to know about it are that it is incredibly low-power and that it reflects light much like normal paper. The technology is currently limited to black-and-white, however.

Why is it important that it reflects light?

Well, it looks and reads just like regular printed text would – you can hold an ePaper display at any angle and still see what's being displayed, whereas with another digital display (like your phone, for example) you have a really limited viewing angle. Because the display reflects light rather than emitting it from behind a glass screen, it's also much more comfortable to read.

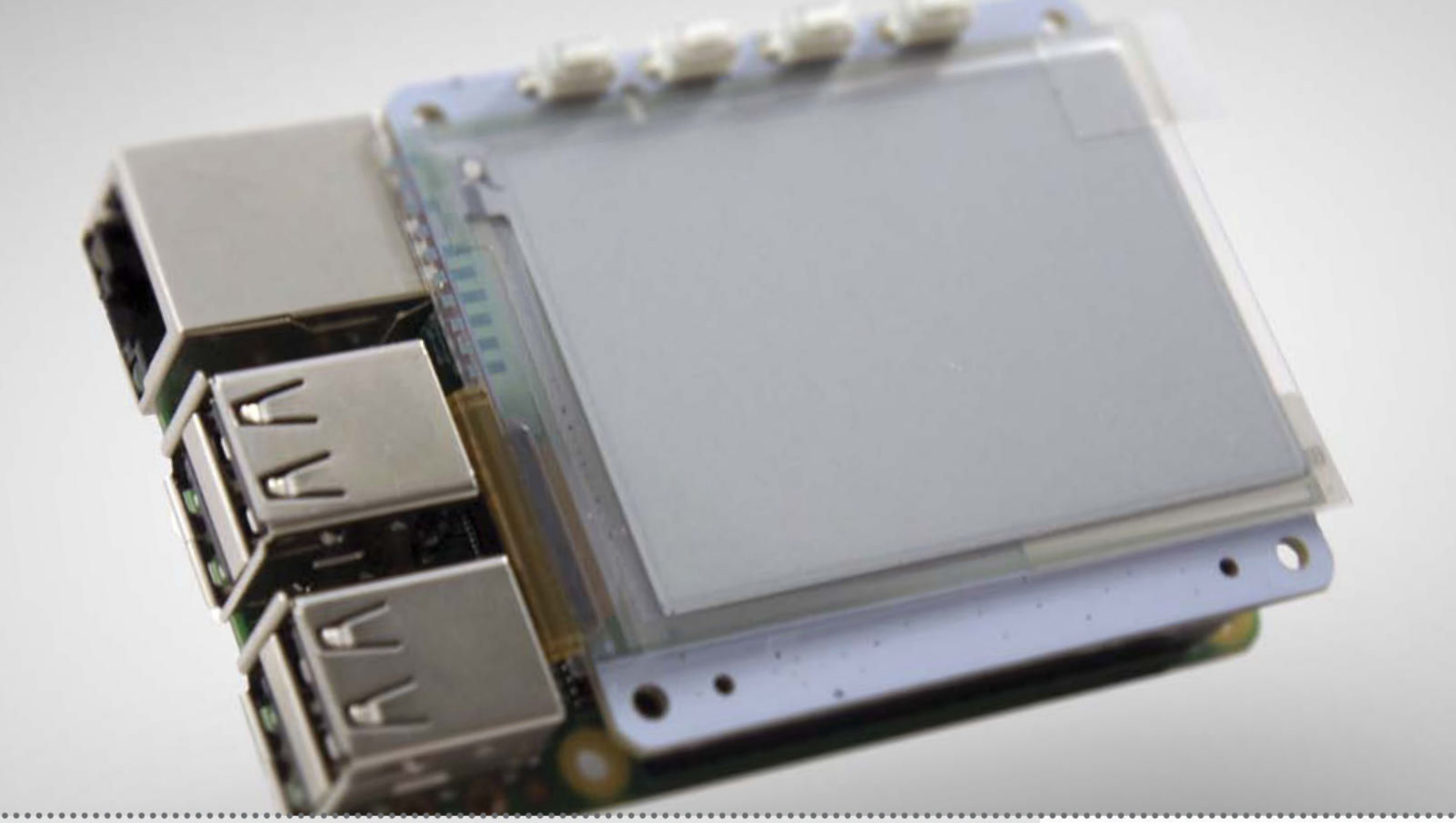
What does the display on ePaper actually look like – what kind of quality does it give?

When the Kindle and Pebble first started using the technology, the quality was around that of a standard printed newspaper. It's improved a little since then but is still very much in the same ballpark. ePaper isn't trying to compete with your smartphone screen, though – again, the key advantage here is that it's incredibly low-power.

Why is that a good thing?

Well, for one thing you can use PaPiRus in any project where you need a permanent display but need to reduce your overheads in terms of power consumption. Imagine you've got a little weather station set up in your garden or some other kind of data logger that needs to display the

"It looks and reads just like regular printed text would – you can hold an ePaper display at any angle and still see what's being displayed"



current readings 24/7, or you're using a Raspberry Pi for digital signage or even a network of them in your shop – cutting down on the wattage is crucial here. Not only that, the PaPiRus can keep an image on-screen for days at a time without receiving any power at all – the image will just slowly fade away.

That sounds amazing! Does the PaPiRus have any other features?

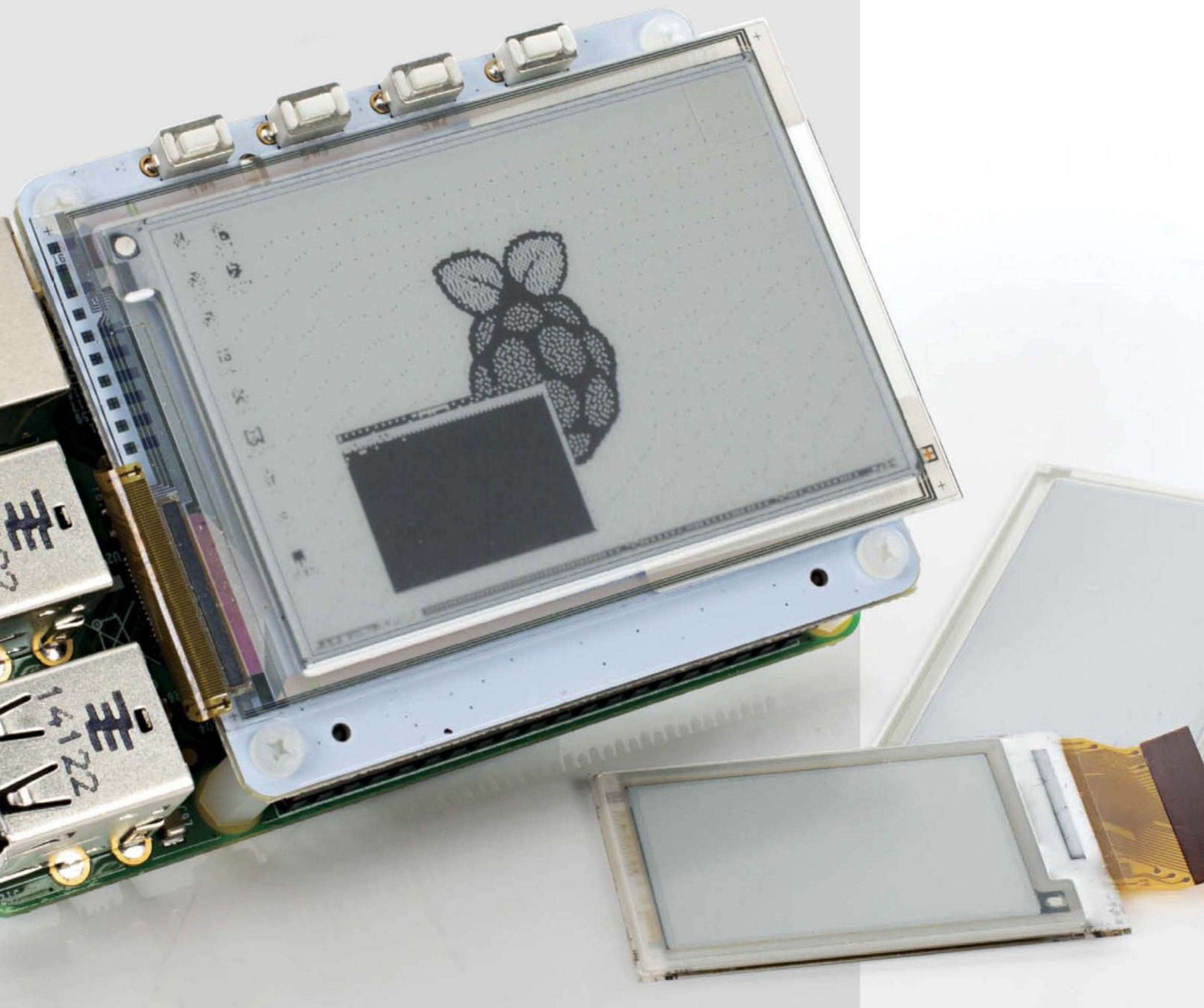
The board itself carries both a temperature sensor and a battery-backed real-time clock. With the latter, you can use the wake-on-alarm functionality to schedule your Pi to switch on at a set time, giving you even more options in terms of permanent project installations. Plus there are four optional switches for the PaPiRus, which you can choose to include when you order one of the modules.

Above As a HAT device, the PaPiRus is made to fit perfectly onto your Pi and also line up with the mounting holes on the board

Okay, I'm almost sold. Just one more question: how big is the display?

The PaPiRus comes with a few different size and resolution options: a 1.44-inch screen (diagonally) with 128 x 96 pixel resolution, a 2-inch screen with 200 x 96 resolution, and a 2.7-inch screen with a 264 x 176 resolution. You can find out all about these options over on the main website, and there should also be wiring diagrams, datasheets and example code available very soon from **www.pi-supply.com**.

"There should also be wiring diagrams, datasheets and example code available very soon"

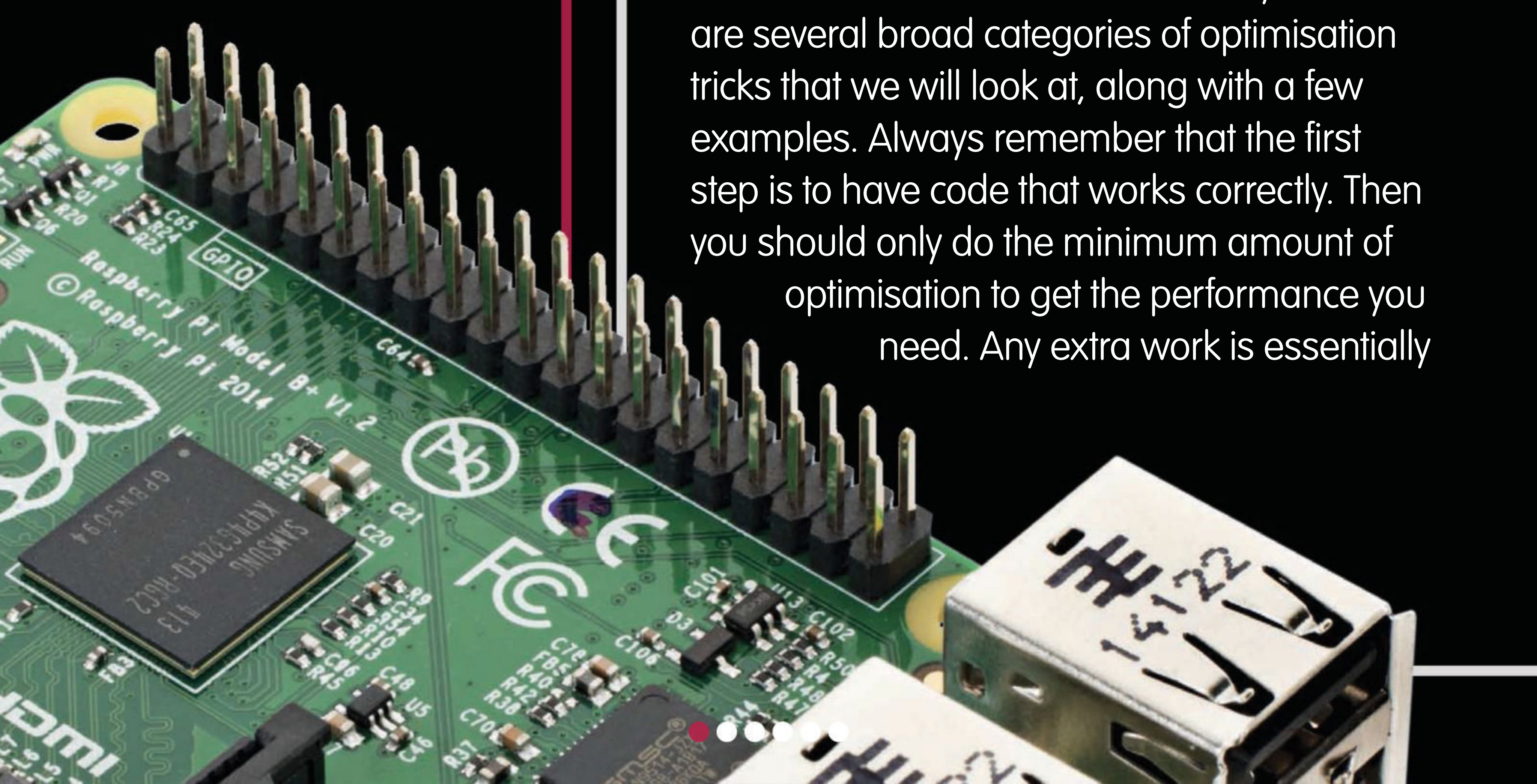


Optimise your profiled Python code

Last month you profiled your code to figure out what needs work. Now let's actually optimise these parts

> "A more Pythonic method is to use classes and objects. You can write a script that defines a class that contains method"

In the last issue, we looked at a few techniques to profile your Python code and figured out which parts were most in need of attention. When you are running code on a Raspberry Pi you have limited resources, hence the need to optimise and get the most work done that you can. This month, we will look at different techniques that can be used to maximise the amount of RAM and CPU that are available to you. There are several broad categories of optimisation tricks that we will look at, along with a few examples. Always remember that the first step is to have code that works correctly. Then you should only do the minimum amount of optimisation to get the performance you need. Any extra work is essentially



wasted, unless you only want to use the process as a learning opportunity.

The first item to look at is how strings are handled. Strings are immutable lists of characters. This means that when you want to add more characters to a starting string, you need to make a new string and copy both the old and new characters into the new space. In those cases where you are programmatically building up large pieces of text, the moving of characters around in memory can be expensive. Instead of using a construct like:

```
str1 = “”
for word in list:
    str1 += word
```

… you can use:

```
str1 = “”.join(list)
```

This builds up the string in a single step, for a reasonable speed-up. Since we are looking at for loops anyway, we should study other things that we can do to make them faster. There is a lot of overhead involved when Python manages a for loop. If the code within it can be bundled as a function, you can use the map command to have this function run for each value stored in a list. In this way, you can remove the overhead of the for loop and move the computational content from Python to C. If you can’t do the shift to using a map command, another item to look at is whether there are references to object methods. If so, Python needs to do type checking on every call around the for loop. This means that Python needs to check the types of the parameters being handed in, then see if there is a

“Strings are immutable lists of characters. This means that when you want to add more characters to a starting string, you need to make a new string and copy both the old and new characters into the new space”

valid form of the called method, and finally run the called method. You can remove this overhead by assigning these calls to a new name before entering the for loop. This way, all of this type checking and name resolution only happens once outside the loop. So the following code:

```
for word in oldlist:
    newlist.append(word)
```

… can be replaced with this more efficient code:

```
append = newlist.append
for word in oldlist:
    append(word)
```

This type of checking also happens whenever you use any kind of polymorphic operator or function. So, if you wanted to use the + operator, Python needs to check both of the parameters being added together to try and figure out what version of + to use. If you do need to use a polymorphic operation inside a very large loop, it might well be worth your time to look at solutions that use strictly defined data types to remove these checks. While this flexibility in data types is a strength of Python, you may need to give up this flexibility in very specific situations to maximise the speed of your code.

If you are doing numerical computation of any kind, you need to start using the NumPy module. NumPy provides functionality that significantly improves the performance of Python numerical code close to the performance you would expect to see when using C or Fortran. The main way NumPy helps is by providing fixed data types, as we mentioned above. The most basic data structure that

NumPy provides is the array. An array is like a list, except that it can only store elements of one type, and it is the basic building block that everything else is built of within NumPy. These arrays are treated as a single data element, so type checks only need to happen once for the entire array. As an example, let's say that you want to multiply the elements of two vectors. In 'regular' Python, you would use a loop like:

```
for index1 in range(50):
    c[index1] = a[index1] * b[index1]
```

… where a and b are vectors of length 50. In this case, every time you run through the loop Python needs to check the types for a[index1] and b[index1]. This is quite a bit of overhead. By using NumPy and the provided array data structures and functions, you can rewrite this code as:

```
c = a * b
```

This makes the code clearer to read. It also only involves a single type check to see whether a and b can be multiplied together. The actual operation then gets passed off to a C library, usually from some linked in linear algebra packages like BLAS or LAPACK. This provides a substantial speed increase because you can take advantage of all of the optimisation work that has gone into these external C or Fortran libraries.

There are some obscure ways of speeding up your code, too. For example, Python needs to regularly check to see whether something else needs to be done. This something else may be looking to see if a different thread needs to run or whether a system call needs to be

"You can take advantage of all of the optimisation work that has gone into external C or Fortran libraries"

processed. This can take quite a bit in terms of resources. You can use the command sys.setcheckinterval() to change how long Python waits to run these checks. If you aren't using threads or making system calls that may be sending signals to your code, you can set this to a higher value to improve the performance of your code. Another more obscure technique that optimises both memory and speed is to use the xrange command. If you need a list of integers, this is usually handled by the command range. The problem with this command is that the entire list gets created in total and stored in memory; you access the list through an iterator and walk through each of the elements. The xrange command provides the list of integers through a generator. A generator does exactly what you think it does; it generates the elements that iterate over as they are needed. This means that you only have one element of the list in memory at any particular time. Also, you don't have the wait while the entire list is being generated. If you are looping over a for loop that has a few thousand cycles, or potentially even millions, this switch to using xrange can provide a significant speed boost. If you are iterating over data in a file, you may actually want to do the opposite. Reading from a hard drive can be much, much slower than reading from memory. This means that you should read in as much of a file as you can reasonably fit into your available RAM in a single operation to speed up manipulating this data. The last obscure method is to consider the scope of variables. Whenever possible, you should use local variables within a function rather than global variables. Python is much more efficient when accessing variables that are within the same scope as the function body, rather than having to move out one or more

layers towards whichever code body actually called the function in question.

If these techniques don't work, you can always move to a different language and use this alternate code within Python. As an example, you can use Cython to take optimised C code and make it available within your Python program. There are equivalent techniques for other languages used in high performance applications. There are several cases, though, where the algorithm available in the alternate language is just not easily translated to efficient code within Python. In these cases, you have the tools available to leverage this previous work within your own code. We will be looking at this particular technique in greater detail next time. The last technique available is to not use Python at all – at least, not the virtual machine. Python is an interpreted language, so every program has the default overhead of having to run through the interpreter rather than running as machine code directly on the hardware. A last resort is to cross-compile your code to machine code for the particular hardware you wish to use. There are a few different projects available, such as Nuitka, that can generate this cross-compiled code. Sometimes, you need to give up portability to get your code running as fast as possible on any particular piece of hardware.

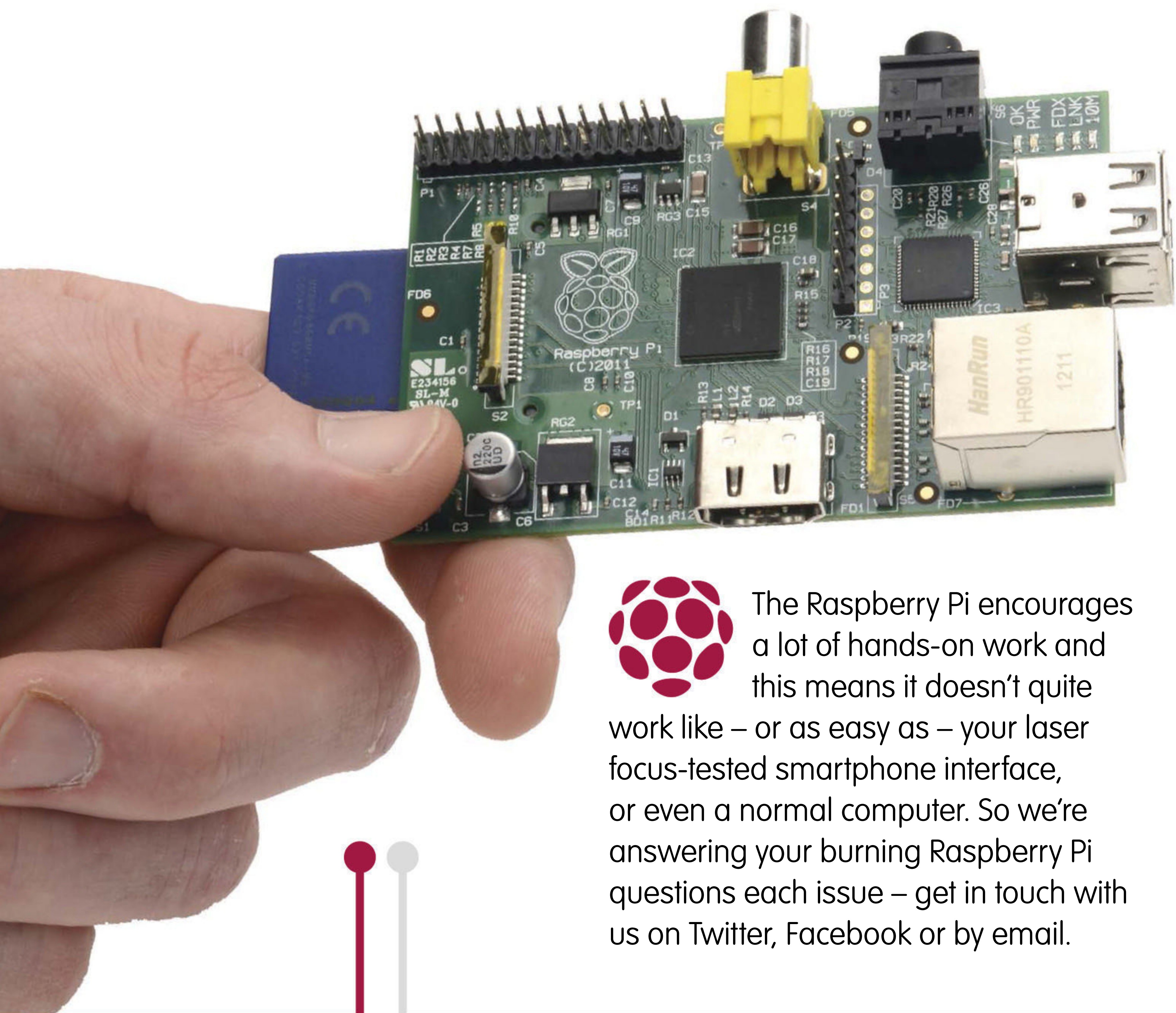
Hopefully these methods will give you the tools to do great things with your Pi. Not long ago, these techniques were common – it was the only way to get work done. The resources available on the Raspberry Pi are limited compared to modern standards. With more thought, much computational work can be done. Just remember the quote, "premature optimisation is the root of all evil," and invest your time where it will have the greatest impact.

"A last resort is to cross-compile your code to machine code for the particular hardware you wish to use. There are a few different projects available, such as Nuitka"

Talking Pi

Join the conversation at…

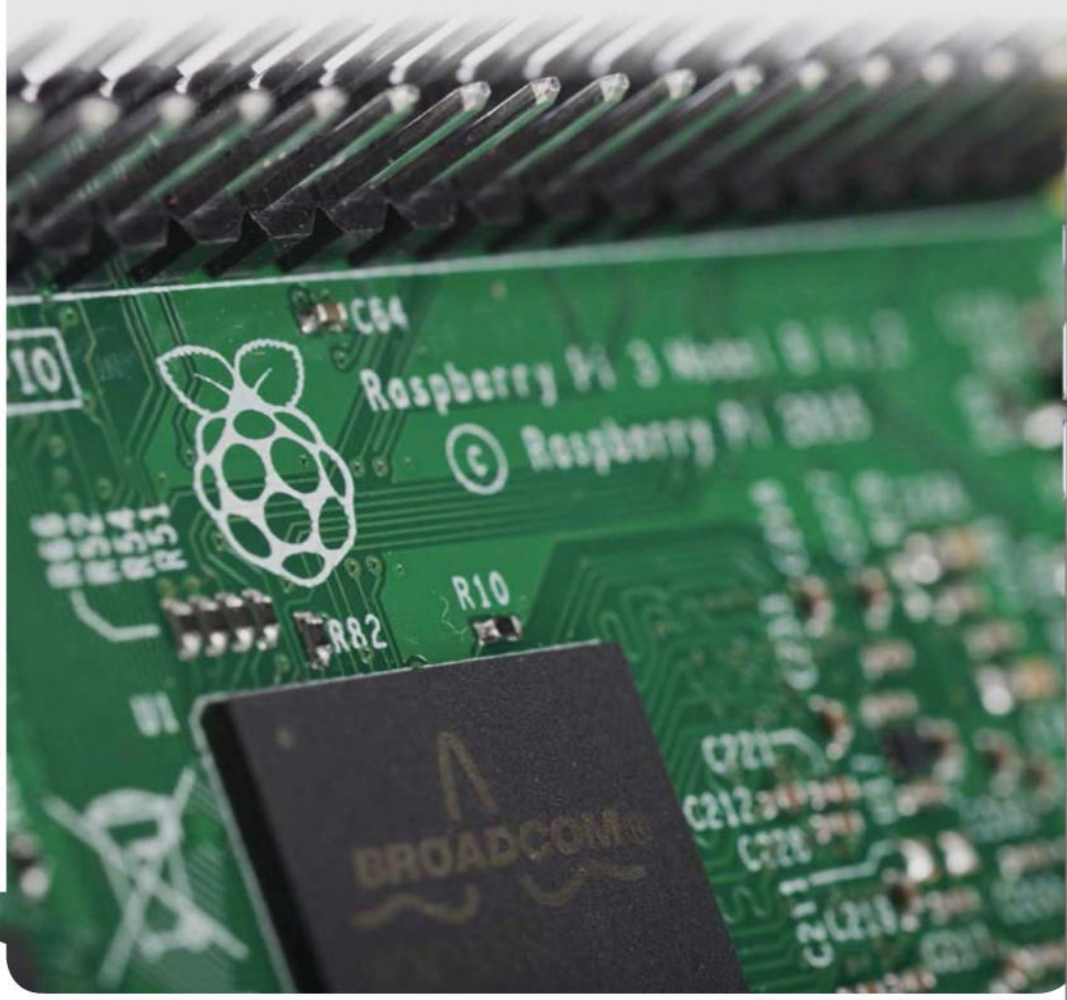
@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk



The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easy as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

Someone on reddit said that the Pi 3 runs way too hot. Does it? How do I prevent it from overheating?
Zach via email

That would be Gareth Halfacree! The renowned hardware tester ran a few benchmark tests on the Raspberry Pi 3 and found that it can reach a temperature of 80°C – even close to 100°C. This is just in very specific circumstances, however, and testing is ongoing to figure out exactly what the conditions are for such a high temperature. It appears to be an issue with CPU throttling, but do keep tabs on the efforts from everyone following the thread: **http://bit.ly/1P8vqds**. In terms of what you can do, ensure that your Pi is well ventilated if you are running intensive tasks, give it downtime whenever possible, and perhaps invest in a small heat sink.



Keep up with the latest Raspberry Pi news by following @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag

JUST A SCORE
WHAT'S YOUR JUST A SCORE?

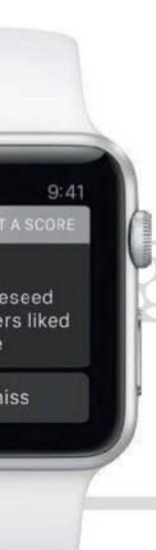
Have you heard of Just A Score? It's a new, completely free app that gives you all the latest review scores. You can score anything in the world, like and share scores, follow scorers for your favourite topics and much more. And it's really good fun!

Will my software setup still work if I upgrade to Pi 3?
Moe via email

It depends on exactly what you've got installed, but generally speaking it should be fine. The Raspberry Pi was designed to enable you to swap out different SD cards, and the Raspberry Pi 3 is fully backwards compatible. So this means that if you stick your current SD card into your new Pi 3, it should boot up absolutely fine – just make sure you shut down the OS on your old Pi properly first. It's also worth checking through any config files you have, to be sure.

How do I get a Wi-Fi signal into my garden?
Sherwin via email

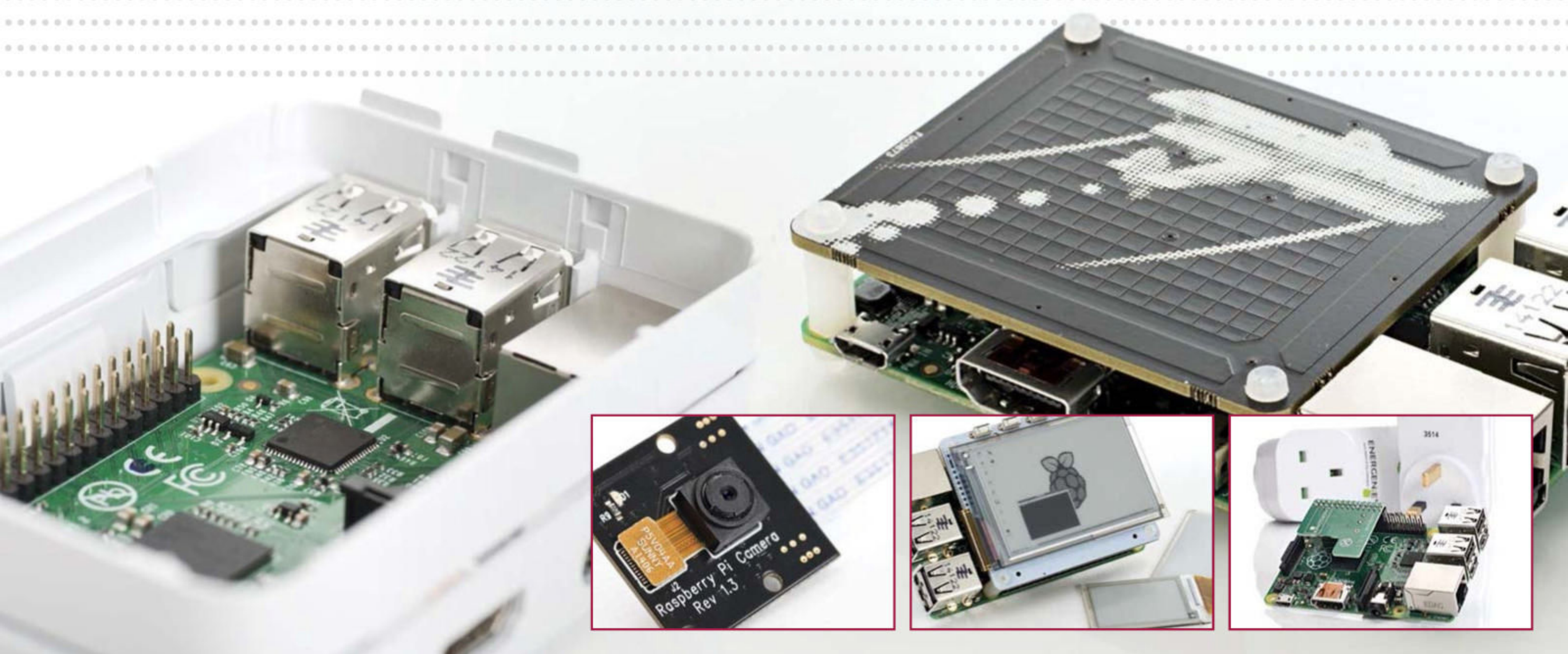
Great question! If you have a Pi 3 then you can take advantage of the on-board Wi-Fi by connecting to the internet via Ethernet, then using the onboard Wi-Fi to broadcast a hotspot for your devices to connect to. Move it out near the back door and see if it reaches. If you don't have a Pi 3, you can just use a USB Wi-Fi adaptor instead. And if it doesn't quite reach down into your garden, you can go wireless by using a portable power pack and then two Wi-Fi modules (or one plus the Pi 3's on-board) – use one module to connect to your home's Wi-Fi router and then the other to broadcast into the garden.



Next issue

Get inspired · Expert advice · Easy-to-follow guides

10 Awesome Upgrades



Get this issue's source code at:

www.linuxuser.co.uk/raspicode